CODING STANDARDS VIOLATIONS IMPACT ON SOFTWARE FAULTS

BY

BASHAR QASEM HEZAM AHMED

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

May, 2014

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN- 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **BASHAR QASEM HEZAM AHMED** under the direction his thesis advisor and approved by his thesis committee, has been presented and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER SCIENCE.**

13/5/14

Dr. Adel F. Ahmed
Department Chairman

Dr. Mahmoud Elish
(Advisor)

Dr. Salam A. Zummo
Dean of Graduate Studies

Dr. Mahmoud Niazi
(Member)

20/5/14
Date

Dr. Mohamed El-Attar
(Member)

إهـــــــــداء

إلى الشمعتين اللتين تنيران دربي ... أمي و أبي العزيزين

إلى من وهبتني كل العطف و الرعاية ... جدتي الفاضلة

إلى ينبوعي الأمل و الصبر ... أم زياد و أم وليد

إلى سر سعادتي و فرحي ... أولادي قرة عيني

DEDICATION

To the two candles lighting my course, my dear parents

To the one who give me the kindness and care, my virtuous grandmother

To the symbols of endless patience and hopefulness, om Ziad and om Waleed

To those who are the secrets of my happiness and joy, my darling kids

ACKNOWLEDGEMENTS

First, and foremost, all praise and thanks are due to Almighty Allah (S.W.T.), the giver of knowledge, for giving me the health, courage and patience to carry out this research. I acknowledge the support given by Taiz University and by KFUPM's Information and Computer Science Department during my graduate studies.

I sincerely and deeply acknowledge with unrestrained appreciation my thesis advisor Dr. Mahmoud Elish for giving me the time and for keeping me focused on my research. I cannot thank him enough for all his patience, encouragement, guidance, support, assistance, constructive feedback. Great thanks are also due to my thesis committee members Dr. Mohammed El-attar and Dr. Mahmoud Niazi for their attention, cooperation, comments and constructive criticism.

I owe a debt of gratitude to my dear parents, the gift of Allah to me. They have always been there for me in my good and difficult times. They have supported me in everything that I have endeavored. Their wisdom and insight have made me into a better person. Words fall short in conveying my gratitude towards them. A prayer is the simplest means I can repay them - May Allah (S.W.T.) give them good health and give me ample opportunity to be of service to them throughout my life. I am also greatly indebted to my family members for their payer and patience. I would like also to thank Mr. Rashad Alyousofi, Mr. Fahd Alhaydri, Mr. Ziad Alrobiah and Mr.Mawal Ali for encouraging and supporting me since the beginning of my graduate study. I would also like to extend my deep thanks for all my colleagues who supported me in many ways and who have been like a second family to me here in KFUPM.

Table of Contents

List of Tables

List of Figures

ABSTRACT

Full Name : BASHAR QASEM HEZAM AHMED

Thesis Title : CODING STANDARDS VIOLATIONS IMPACT ON SOFTWARE FAULTS

Major Field : COMPUTER SCIENCE

Date of Degree : MAY 2014

Software quality has been a subject of extensive research in the last few decades. One special area of research in software quality is the relationship between software quality and coding standards. During the last few years, more attention has been paid to the impact of coding standards on software quality attributes. Many quality attributes has been addressed such as maintainability, stability and reliability. Functional correctness in terms of fault proneness and fault density is one of the most important software quality attributes that needs to be addressed extensively by more researches. Also many coding standards have been proposed for the sake of improving the software quality through enforcing such standards during writing code lists. Better understanding of the relationship between coding standards violations at the class level and software faults is an important software engineering issue as it helps to predict faults and thus mitigate them, target the available resources more effectively, and identify the problematic parts of the software system. Coding standards violations-based metrics, which are the means to quantify the coding standards' rules violations, are potentially good indictors of the fault proneness and fault density in the software system.

The objective of this research is to derive and empirically validate a set of coding standards violations-based metrics as potential indicators of the fault-proneness and the

fault density of a class of object-oriented system. Two families of statistical prediction models (univariate and multivariate regression) are built. The multivariate models are built in three different ways: using CK metrics, using the coding standards violations-based metrics and using a combination of both, as predictors for both fault proneness and faults density.

The results indicate that the coding standards violations-based metrics are measuring different dimensions from those of the CK metrics. Additionally, several coding standards violations-based metrics were found to be correlated with both fault proneness and fault density of classes. Moreover, the results showed that more accurate prediction of class fault-proneness is achieved when the coding standards violations based metrics are combined with CK metrics.

ملخص الرسالة

الاسم الكامل : بشار قاسم حزام أحمد.

عنوان الرسالة: تأثير إنتهاك معايير كتابة الكود على أخطاء المنتجات البرمجية.

التخصص : علوم الحاسب الآلي.

تاريخ التخرج : مايو 2014 م.

استمرت جودة المنتجات البرمجية موضوعاً للبحث المستفيض لعقود من الزمن. و كانت دراسة العلاقة بين جودة المنتجات البرمجية و معايير كتابة الكود إحدى أهم مواضيع ذلك البحث لا سيما في السنوات الأخيرة فقد تم إعطائها المزيد من الإهتمام. كما تمت خلال الفترة القليلة الماضية دراسة العلاقة بين إنتهاك معايير كتابة الكود و بعض خصائص جودة المنتج البرمجي إلا أن أهم هذه الخصائص لم تُدرس بشكل كافي و هي خاصية خلو المنتج البرمجي من الأخطاء. هذا و يعتبر فهم العلاقة بين إنتهاك معايير كتابة الكود و تواجد الأخطاء في المنتجات البرمجية قضية مهمة في هندسة البرمجيات و ذلك لأنها ستساعد في إدارة و توجيه المصادر بشكل فعال و ستساعد أيضاً في تحديد الأجزاء المعطوبة في المنتجات البرمجية. و لتقييم تلك العلاقة فإنه لابد من إستخدام وسيلة لتمثيل الإنتهاكات لقواعد كتابة الكود بشكل كمي يسهل معه التعامل مع بيانات تلك الإنتهاكات. و هنا تم استخدام المقاييس المبنية على هذه الإنتهاكات لتمثيلها كمياً و بالتالي يصبح تقييم تلك العلاقة أمراً ممكناً و منطقياً.

في هذا البحث تم إقتراح المقاييس المبنية على الإنتهاكات لقواعد كتابة الكود لتستخدم كمؤشرات لتحديد الأجزاء المعطوبة من المنتج البرمجي و كذلك تحديد حجم العطب أو كثافته في المنتج البرمجي. إن هذا البحث يهدف بشكل أساسي إلى: 1) بناء نوعين من النماذج الإحصائية (نماذج وحيدة أو منفردة و نماذج متعددة). و قد تم بناء النماذج الإحصائية المتعددة بثلاث طرق مختلفة: إستخدام كلاً من المقاييس المبنية على إنتهاكات قواعد الكود إضافة إلى مقاييس المنتج, إستخدام المقاييس المبنية على الإنتهاكات فقط, إستخدام مقاييس المنتج فقط. 2) التحقق من هذه النماذج من خلال تصميم تجارب عملية على أكثر من برنامج من البرمجيات مفتوحة المصدر.

لقد أظهرت نتائج البحث أن المقاييس المبنية على الإنتهاكات نفسها تقييس نواحي مختلفة عن تلك التي تقيسها مقاييس المنتج. كما أظهرت النتائج أن كثير من المقاييس المبنية على الإنتهاكات و التي تم إقتراحها في هذا البحث لها علاقة أكيدة بالعطب و حجم العطب أو كثافته في المنتج البرمجي. إضافةً إلى ذلك, أظهرت النتائج بأن أفضل النماذج التوقعية دقةً هي تلك التي تم بناءها بإستخدام المقاييس المبنية على الإنتهاكات إضافةً إلى المقاييس المستشفة من الخصائص الهيكلية للمنتج البرمجي.

CHAPTER 1

INTRODUCTION

Coding standards or programming styles refer to the set of rules that are shared internally among software development team members and emphasized or enforced by their projects' managers. "These rules in such standards are typically based on expert's opinions, and can be targeted towards multiple quality aspects, such as reliability, portability or maintainability" [1]. Programming styles and coding standard's rules are designed to reflect different concerns and affect areas of source code writing, with the aim of improving the readability of source code and hence improving the maintainability. Furthermore, coding standards control or even prevent the usage of the known faulty constructs of a programming language to reduce the software bugs of the underlying software systems. Such coding standards range from language-independent typographic styles in which the rules affecting how both the source code and comments are visually structured and displayed [2, 3], to general programming practices relative to specific programming languages such as C++[4], paradigms such as the object-oriented paradigm [5] or even development approaches such as the Agile methodologies[1].

The presence of coding standards and programming styles has already confirmed by previous researches in a wide variety of approaches such as commercial and proprietary software [6] and also in agile-driven software systems [7] as well as in OSS [8]. It can be observed that also the previous empirical research works have shown that applying coding standards either during the creation of new code or maintaining existing

code has a great impact on the quality of the resulting software product [3]. For instance, Sun argues that its internal java coding conventions involved in java coding standard 1999 [9] help new developers to more easily understand existing code and reduce maintenance costs. This Standard is considered one of the most well-known and widely used standards by the java community.

"Almost all software contains defects" [10]. The detection difficulty for those defects is found to be easy for some of defects and very difficult for other defects due to their absence or seldom emergence. Some of these defects emerge relatively often, however, they go unnoticed because their lower severity or because they are not realized as errors. Software defects range from functional defects in which the program computes incorrect values to runtime defects in which the program typically crashes, or resources leaks defects in which the program's performance degrades and reaches the frozen state [10].

Some parts of the software may be more prone to faults than others and, as implied by Pareto's law, 80% of problems, such as defects, changes, rework, and so on, are rooted to only 20% of the classes in a software system. This phenomenon has been empirically validated by Porter and Selby [11], and Kuro and Liu [12] on both commercial and open-source systems respectively. Therefore, identifying and characterizing which classes are fault-prone can be very useful and helpful in guiding the maintenance team and distributing the resources more efficiently. Consequently, this will enable the project manager and his team to focus their effort and attention on the fault-prone classes of their software systems [12].

Software quality can be defined in terms of quality attributes. One of these attributes is the functional correctness that can be defined as the degree to which a product or a system provides the correct results with the needed degree of precision. Functional correctness is a sub-attribute of functional suitability according to the ISO 25010 standard. Functional correctness can be measured in terms of fault proneness and fault density which in turn, can be linked to coding standards violations as discussed in the next chapter. Fault proneness, that is, whether a class is faulty class or not. Fault density can be defined as the number of confirmed faults detected in software/component during a defined period of development/operation divided by the size of the software/component.

1.1 Technical Background

To build software systems, which are reliable, scalable and maintainable, it is important for development teams to adopt proven design techniques and good coding standards. Such adoption of coding standards will eventually results in code consistency, which in turn makes the software easier to understand, develop and maintain. Additionally, by being aware of and following the right coding techniques at a certain granular level, the programmer can make the code more efficient and performance effective as stated in recommendations 26 and 34 of European space agency's coding standard [13].

Furthermore, coding standard's rules can be designed to affect different software quality attribute[1]. The functional correctness is one of those quality attributes. In this respect, an extensive research works have to be done to investigate the potential impacts

of applying such standards on different software quality attributes. One useful technique to achieve such investigation is the usage of software measurements.

It has been a main objective in science to come up with a way to quantify observations for the sake of easily understanding and controlling the underlying issues [14]. In this regard, software engineering has also employed the use of quantified observations data in both forms quantitative and qualitative to improve the software products and thus come up with software products in a predictable cost, schedule and quality.

Software measurements have existed since the first compiler counted the number of lines in a program listing [15]. Several quality aspects of software are characterized using metrics. Thus, software metrics are the instrument for applying Edwards Deming's (1900 - 1993) advice: "You can't manage what you can't measure". Fenton [16] in his book broadly classified software metrics into three main categories: product, process, and resource metrics. Product metrics are those that describe characteristics of the software development life cycle processes outputs. They are measures of the software at any stage of its development and maintenance. Examples of such metrics are size, coupling, and cohesion metrics. Process metrics measure attributes related to the software development life cycle processes in order to improve them. The most relevant processes attributes are time, effort and cost. Resource metrics describe the available resources characteristics such as the number of developers, development environment level and hardware performance metrics.

Furthermore, software metrics can be classified into internal and external metrics. Internal metrics can be measured in term of the entity itself. Examples of these metrics

are size, communication level and effort. While external metrics are those that can be measured only with respect to how the entity relates to its environment. They are less tangible than the internal ones, and they assess the external characteristics of the entity, like cognitive complexity, maintainability, quality and understandability. Software metrics can be classified into static and dynamic metrics as well. Static metrics are those collected from the static artifacts of software, such as specification documents, design diagrams and code listings. Examples of static metrics include Lines of Code (LOC), Weighted Methods per Class (WMC) and Coupling between Objects (CBO). On the other hand, dynamic metrics are collected during the run time of the software from its executable form. Extent of class usage, Dynamic Coupling, and Dynamic Lack of Cohesion are example of this type of metrics. [16]

Software metrics quantitatively provide useful information that can help in many ways: assessing software product and process, providing feedback to engineers, and guiding project managers in decision making. Software metrics can be elicited from different aspects of the software systems. For instance, the class cohesion metric can be computed from interactions between the different components of the class. In our case, coding standards violations metrics are derived from the violations of coding standards rules in programs coding artifacts. For example, number of violations in each statement, method, class or system as a whole can be used as metrics in order to study some attributes which are believed to participate in improving software quality.

This research work main focuses on the static coding standards violations-based metrics. In this regards, this research also proposed a set of coding standards violations-based metrics which can be computed from the source code artifacts of the open source

software projects under study by means called static analyzers. The percentage of coding standard's rules being violated in the class, The percentage of coding standard's categories being violated in the class, etc, are examples of such metrics. All those metrics are new and can be adapted to be used with any coding standard. Detailed definition and description of these metrics is given in Section 4.1.1.

This research work chose to compare the proposed coding standards violations-based metrics with C&K metrics [17] due to the following. C&K metrics is a well-defined suite of OO metrics in literature. They have been theoretically validated. Also they have been empirically investigated and found to be associated with different quality aspects. Additionally, they measure different structural properties of the system like size, coupling, cohesion, and inheritance. Furthermore, many tools support them. Definition and detailed description about this suite of metrics is presented in Section 4.2.1.

1.2 Research Problem

The usage of coding standards and the tools enforcing their rules is becoming a popular trend in software development especially during the writing of code lists [18]. Coding standard's rules can be targeted towards different software quality attributes and hence are believed to improve quality [1]. However, there is no empirical evidence on the relationship between coding standard's rules violations at the class level and the presence of faults and their density. Empirically, a repertoire of Statistical models has to be built using the proposed coding standards violations-based metrics to identify fault-prone classes and estimate the fault density of these classes. Also a repertoire of statistical models has to be built using the existing product (CK) metrics for the sake of comparing

the performance of the proposed metrics with the performance of CK metrics. So this research work aims to answer these questions:

- *Does the violation of coding standard's rules has a relationship with the existence of faults in software products at the class level?*
- *Compared to the CK metrics, how accurate are the coding standards violations-based metrics in identifying the fault-prone classes and in estimating the fault density of these classes?*
- *Having a combined prediction model based on both the coding standards violations-based and the CK metrics, how accurate is this model, compared to a model built only from one of them?*

1.3 Research Objectives

The main objectives of this research are:

- Propose a set of coding standard violations-based metrics.
- Empirically explore the relationships between coding standard's violations and fault proneness at the class-level.
- Empirically explore the relationships between coding standard's violations and fault density at the class-level.
- Use the proposed coding standard's violations metrics to build fault prediction models and evaluate their prediction power.

1.4 Research Motivation

Coding standards or programming guidelines are becoming more popular as means for ensuring software quality during the development process. Those standards

intuitively improve the overall quality of the software products because their rules prevent or control the usage of the known problematic or faulty constructs of a programming language [18]. Since faults-proneness and fault density are important attributes of software quality, it is reasonable to study the factors that potentially participate in increasing faults in software products. One potential factor is the violations of coding standard's rules during source code writing. Since there were no empirical evidence about the relationship between coding standard's rules violations and faults at the class level, this research aims to address such relationship to come up with recommendations for software development teams' managers to adopt such standards and ensure the compliance for coding standards by their team's members during source code writing of the software projects. It can provide some sort of ranking for coding standard's rules severity as well. The evaluation of coding standards violations and the functional correctness in terms of fault proneness and fault density can also help in providing further insights for understanding software quality.

1.5 Research Contributions

The contributions of this research are as follows:

- Surveying the literature for identifying the existing coding standards violations-based metrics for evaluating software quality attributes.
- Proposing and empirically validating a set of coding standards violations-based metrics for improving object-oriented software faults prediction, in terms of whether a class is fault-prone or not and the faults density of classes.
- Building and evaluating fault prediction models for classifying classes into fault-prone and non-fault-prone classes for object-oriented software.

- Building and evaluating fault prediction models for estimating fault density of OO classes.
- Comparing the performance of product and coding standards violations-based metrics as predictors for fault-proneness of the classes of object-oriented systems.
- Comparing the performance of product and coding standards violations-based metrics as predictors for the fault density of the classes of object-oriented systems.
- Identifying which coding rules/categories have impacts on software faults.

1.6 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 summarizes the literature. Chapter 3 presents the coding standards and static analysis in details. The suites of metrics investigated by this study are presented in Chapter 4. Chapter 5 presents the coding rules violations database design. Chapter 6 and 7 present the empirical study design, results and analysis. Chapter 8 concludes the research work and presents the future work.

CHAPTER 2

LITERATURE REVIEW

Many research works have been done in the literature that addressed different quality attributes and different coding standards. Some of these researches addressed the relationship between well publicized coding standards and software quality attributes, while others addressed such relationship between self-imposed or industrial coding standards and software quality attributes

2.1 Coding Standards Violations-Based Metrics and Quality Attributes

Since this research work addressed the functional correctness in terms fault proneness and faults density, the research works that have done in the literature are summarized into two sections. Section 2.1.1 summarized the previous works in which the fault proneness and fault density are addressed while Section 2.1.2 summarized the previous works in which other quality attributes are addressed.

2.1.1 Coding standards Violations-Based Metrics and Software Faults

Moonen and Boogerd [18] applied the MISRA-C: 2004 [19] coding standard to measure the quality of source code of two closed commercial projects before and after bug fixes during the development of two closed source embedded C applications. They propose simple metric called violations density which is the number of violations divided by the number of lines of code of the corresponding unit (project, module, and file). They considered 89 coding rules belonging to different coding categories. As a result, they

found that only 10 rules from the considered 89 rules are significant predictors for faults locations. Those 10 rules were found to be positively correlated with fault proneness.

In another work, Moonen and Boogred [1] applied the MISRA-C:2004 [19] coding standard against all the revisions of two closed commercial source projects. To build a body of empirical knowledge to understand the relationship between coding standard's violations and faults density, they used two metrics called violations density metric (the number of violations per version divided by the number of KLOC for that version) and fault density metric (the number of faults per version divided by the number of KLOC for that version) at the system level. Their study considered only 72 rules out of 141 rules of MISRA-C 2004 standard. As a result of their study, they found that there is a positive correlation between violations density and faults density only for 12 rules.

Basalaj and Beuken [20] used coding standard's violations metric as a measure of internal quality of software source code. Their study tested the number of coding guidelines violations metric against 18 closed source products written in C and C++ of two software production companies. Among the 900 rules of high-integrity C++ [21], MISRA-C [19], they found a positive correlation between coding rules' violations and faults only for 12 rules out of the mentioned 900 rules. In addition to faults they also found that the compliance to a coding standard has been found to positively impact the portability of the software products.

In their study, Kawamoto and Mizuno [22] evaluated the relationship between the length of identifiers and the existence of software faults in a software module. To investigate such relation, they built a model to determine faulty-module using a machine learning technique from the number of occurrences of the identifiers. Their study tested

two metrics *Oc(l)* which is the number of the occurrences of identifiers with length *l* in a module (they considered the length of the identifier as one of the Characteristics of identifier's naming rules) and *TN* which is the total number of identifiers found in a module against two open source projects. As a result for their experimentation, they showed that there is a certain relationship between the length of identifier and the existence of software faults and they also specified the best length the identifiers should have.

2.1.2 Coding standards Violations-Based Metrics and Other Quality Attributes

In their study, Takai et al [23] evaluated the relationship between the code size (LOC) at the granular level of the system as a whole and the number of coding standard violations. They proposed three metrics based on the number of violations. Those metrics are as follows, NOV which is the number of coding standards violations, CNV which is the cumulative number of coding standard's violations and CNUV which is the cumulative number of unique coding standard's violations. Their proposed metrics are used against 127 rules from the MISRA-C 1998 coding standard. As a result, they found positive correlation between the code size and the number of coding rules violations.

Elish and Offutt [24] conducted a controlled small-scale experiment that tries to determine to which extent the open source java programmers adhere to a set of well publicized coding practices. They evaluated in their experiment sixteen coding standard practices or rules. It is worth here to mention that their selection for those rules was according to two predefined criteria: (1) they can be checked automatically using static checkers tools, and (2) they are widely used by the java development community. They selected one hundred classes arbitrarily from random open source projects to be

examined against those sixteen rules which are grouped into four categories: visibility, formatting, control structures and naming convention. To measure the developer's adherence for the selected rules or practices, they used the number of distinct violations of rules at the class-level. As a result of their study, they found that only five of the sixteen rules or standard practices are followed by all subjects. Additionally, they found that only 4% of the subject classes have no violations. They found that 90% of the subject classes have at most 4 violations as well. Also they found that there are positive correlation between the size of the class in terms of LOC and the number of violations found in that class.

Smit et al [25] analyzed the potential of code convention adherence as a predictor of maintainability. They systematically examined all of the revisions of four open source java projects, checking for adherence to their self-imposed standards and to a set of sun coding conventions specified by a panel of software engineering experts to be important to maintainability. To support their process they developed a set of tools to automate and visualize the process. Their study evaluates the number of violations for specific rule i.e. counting the number of violations for specific rule of the corresponding standard for each project at the granular level of system as a whole. Each project is tested against exactly 71 rules which are considered as important to write maintainable code. As a result of their study, they found that developers had better adherence to self-imposed standards but did not have better adherence to those rules identified by the expert panel. They also found that, the number of violations grows in a linear relationship with the length of code. Furthermore, they found that there is evidence which support their suggestion that the number of violations has bearing on the maintainability of a software projects.

In another study, Smit et al [26] examined four open source projects against 32 code conventions from the sun coding standard. They studied the relationship between the number of violations at the granular level of a system as a whole in terms of system commits and the number of contributors to the open source system. They also studied the relationship between the number of violations and the size in terms of KLOC. As a result, they found that the number of violations increased linearly with the size of code and they also found a positive correlation between the number of contributors and the number of violations introduced to the system.

Butler et al. [27] study the extent to which the quality of identifier names might influence the quality of source code. They studied the relationship between the quality of identifier names and the readability at the granular level of java methods. Furthermore, they studied the relationship between the quality of identifier names and FindBugs warnings. They investigated 8 open source java applications using 10 identifier flaws. As result of their study, they found that poor quality identifier names are strongly associated with more-complex, less-readable and less maintainable source code. They also found that poor quality identifier names are strongly correlated with FindBugs warnings; however, the relationships are complex and appear to be application-specific.

2.2 Literature Summary and Comparisons

To sum up, Table 2.1 gives deeper insight into the differences between this study and what have been done in the literature. Major number of those studies has focused on the highest granularity level which is the software system as a whole in terms of its releases. This makes it difficult to identify which portion of the software system needs to be reviewed or refactored.

Moreover, even in those studies that have used the coding standards violations-based metrics on the class level, the researchers used them in a limited way. For example in [24], they conducted a controlled small-scale experiment that tries to determine to which extent the open source java programmers adhere to a small set of coding practices. Similarly, in [22] they used, as coding standards violations-based metrics, only one metric called The number of occurrences of identifiers with length *l* in a class which collect the violations for only one rule related to the naming conventions.

Additionally, the target set of systems under study of all the previous studies was small which in turn restrict the generalization of the obtained results. Even in [20], although they used 18 closed source products in their study, they used only one metric which is the number of coding standard's violations per software product in terms of versions which in turn, makes the prediction models unsatisfactory.

Unlike the previous ones, this research work is different in many aspects. First of all, as far as we know, the relationship between coding standards violations and faults has not been sufficiently studied on the class level, whereas in this research work, this relationship is addressed on the class level. Secondly, in addition to some of the already proposed metrics, new set of coding standard violations-based metrics are proposed. Finally, in the previous studies, the targeted set of systems was small which make it unsafe to generalize the obtained results, but the target set of systems investigated by this study is big enough to generalize the obtained results and to draw clear conclusion safely. All those aspects enable one to make benefits from the obtained results. Furthermore, this work enables one to identify problematic parts of the software earlier in the development

life cycle. Finally, identifying which rules of the adopted standard should be enforced and followed to the extreme is good example of the possible gained benefits.

Table 2.1 Comparison between the proposed work and the literature work

Research Study	Subject Systems	Programming Language and Granularity Level	Coding Standard	Coding Standards Based Metrics	Empirically evaluated (Quality Attribute)
This research work	Xalan 2.6.0, Velocity 1.6.1, Synapse 1.2, Poi 3.0, Ant-1.7.0, camel-1.6.0	Java (class-level)	JPL coding standard (43 Rules)	6	yes (Fault-prone and Fault Density)
Elish and Offutt [24]	100 classes from arbitrary open source projects	Java (Class-level)	Sun coding standard (16 Rules)	1	no
Basalaj and Beuken [20]	18 closed source products	C++ (System-level)	MISRA-C 1998,2004 and high-integrity C++ (900 Rules)	1	yes (Faults)
Moonen and Boogerd [1]	TV on mobile, NXP TV platform (TVC)	C (System-level)	MISRA-C 2004 (72 Rules)	1	yes (Fault Density)
Moonen and Boogerd [18]	TV on mobile, NXP TV platform (TVC)	C (System, module, and file)	MISRA-C 2004 (89 Rules)	1	yes (Fault-prone)
Smit et al [26]	Ant, Derby, Hadoop, Jfreechart	Java (System-level)	Sun coding Standard (32 Rules)	1	no
Takai et al[23]	TOPPERS/ASP Kernal, runvpn	C and C++ (System-level)	MISRA-C 1998 (127 Rules)	3	no

Smit et al[25]	Ant, Derby, Hadoop, Jfreechart	Java (System-level)	Sun coding standard + self-imposed standards (71 Rules)	1	yes (Maintainability)
Kawamoto and Mizuno [22]	Eclipse and Netbeans	Java (Class-level)	The length of the identifiers (1 Rule)	1	yes (Fault-prone)
Butler et al [27]	8 open source projects	Java (Methods-level)	Naming conventions (10 rules)	1	Readability

Table 2.2 presents the proposed and used metrics that we are going to use in this study in addition to the already existing metrics which have been proposed and used in the previous work.

Table 2.2 The proposed metrics by this research and the proposed metrics in the literature

Research Study	# of Metrics	Metric Appreciation	Metric Definition
This research	6	PSRV	The percentage of JPL standard's rules being violated per class.
		PSRVD	The percentage of JPL standard's rules being violated in the class which is then normalized by the class size.
		PNCRV	The percentage of names category rules being violated in the class.
		PNCRVD	The percentage of names category rules being violated in the class which is then normalized by the class code size.
		PPCICRV	The percentage of packages, classes and interfaces category rules being violated in the class.
		PPCICRVD	The percentage of packages, classes and interfaces category rules being violated in the class which is then normalized by the class code size.
		PFCRV	The percentage of fields category rules being violated in the class.

		PFCRVD	The percentage of fields category rules being violated in the class which is then normalized by the class code size.
		PMCRV	The percentage of methods category rules being violated in the class.
		PMCRVD	The percentage of methods category rules being violated in the class which is then normalized by the class code size.
		PDSCRV	The percentage of declarations and statements category rules being violated in the class.
		PDSCRVD	The percentage of declarations and statements category rules being violated in the class which is then normalized by the class code size.
		PExpCRV	The percentage of expressions category rules being violated in the class.
		PExpCRVD	The percentage of expressions category rules being violated in the class which is then normalized by the class code size
		PExcCRV	The percentage of exceptions category rules being violated in the class.
		PExcCRVD	The percentage of exceptions category rules being violated in the class which is then normalized by the class code size.
		PTCRV	The percentage of types category rules being violated in the class.
		PTCRVD	The percentage of types category rules being violated in the class which is then normalized by the class code size.
		PConCRV	The percentage of concurrency category rules being violated in the class.
		PConCRVD	The percentage of concurrency category rules being violated in the class which is then normalized by the class code size.
		PComCRV	The percentage of complexity category rules being violated in the class.
		PComCRVD	The percentage of complexity category rules being violated in the class which is then normalized by the class code size.
		PSCV	The percentage of categories that have been violated in the class.
		PSCVD	The percentage of categories that have been violated in the class which is then normalized by the class size.
Elish and Offutt[24]	1	--	The number of violations per class.
Basalaj	1	--	The number of coding standards violations.

and Beuken [20]			
Moonen and Boogerd [1]	1	--	The Violations density.
Moonen and Boogerd [18]	1	--	The violations density.
Smit et al [26]	1	--	The cached convention violations.
Takai et al[23]	3	NOV	The number of violations.
		CNV	The cumulative number of coding standards violations.
		CNUV	The cumulative number of unique coding standards violations.
Smit et al [25]	1	--	The Number of violations per rule
Kawamoto and Mizuno [22]	1	*OC(l)*	The number of occurrences of identifiers with length *l* in a module.
Butler et al [27]	1	--	The number of violations.

CHAPTER 3

CODING STANDARDS AND STATIC ANALYSIS

3.1 Coding Standards

Coding standards and programming styles form a set of pre-defined formal rules which are internally shared among software project team members, and enforced by software projects managers by applying static analysis during the source code writing of the software development [6]. The rules of these standards are typically based on expert's opinions, and can reflect different concerns and affect different aspects of source code writing with the aim of improving many quality attributes of the underlying software system [1].

Nowadays, increasingly more emphasis is given to encourage developers adhering to such coding standards and practices and hence preventing them from using language constructs that are known to be potentially problematic, especially for high cost of failure software projects [18]. For example, sun developed and published its coding standard for java programming language and argued that its standard help software developers to understand the existing code easily and also to reduce the software maintenance cost [1]. Another example is the MISRA C standard [19] which is intended specifically to control the use of C programming language constructs in real time and critical systems. Coding standards have co-evolved with programming languages and some standards are generally applicable while others are specific to individual programming languages like Java or dedicated to a paradigm like OOP.

In addition to the well-publicized, well known and widely used standards by the software community, there are many other self-imposed or commercial standards developed and owned by large software development organizations which produce software components to be adhered by software engineers across their software projects. To meet the fact that everybody will be eventually in everybody else's code, software production companies enforce those standards to ensure that the transition of code to new developers will goes on smoothly. Furthermore, those companies emphasis and enforce such standards to ease the transition of developers from team to another.

Coding standard term usually used as a broad umbrella that includes almost all best practices connected to the process of writing code lists. For instance, the European Space Agency's standard for java programming language [13] categorizes all coding practices into several categories, (1) Installation, Build and Updates category in which, they put rules and recommendations that provide common and uniform build and update procedures to allow convenient and reliable deployment and installation of software products. (2) Source Code Structure category in which they involve rules and recommendations that define a consistent code formatting style to be applied in their projects for the sake of meeting the fact that uniform source code structure and formatting is fundamental for an adequate collaboration between programmers. (3) Naming Category contains rules and recommendations that help in choosing descriptive program identifiers in order to keep the produced code clear and self-documenting. (4) Documentation and Commenting Conventions Category which includes rules and recommendations that encourage developers to write additional explanatory documentation and comments with their produced code to meet the fact that the final and

most reliable source of information about a project is its own source code which is often difficult to be interpreted in isolation of its documentation. (5) Java Design and Programming Guidelines category that comprises rules and recommendations formulated from the lessons accumulated during the long time of software development with the Java programming language as well as with other object oriented languages. (6) Robustness Category's rules and recommendations help in instrumenting code in such a way so that the produced code can detect and correct errors especially during execution. Some of this category rules and recommendations are directed to achieve more robust code through the design by contract and assertions techniques, while the others, particularly those related to error handling, are more concerned with run-time robustness. (7) Portability Category includes rules and recommendations which are intended to support programmers in producing 100% portable Java code especially for those applications that cannot be (and probably do not need to be) 100% portable, for example applications that access hardware and Code written for real-time Java virtual machines. Even in such cases, however, the rules and recommendations included in this category could be helpful, since achieving a maximum of portability is always valuable. (8) Real-Time Java Category's rules and recommendations are oriented towards making the use of Java for real-time systems implementation as effective and reliable as possible.

In order to write great software, you have to write software greatly. The point is that before you can produce great code, you have to adopt proven design techniques and good process for writing consistent and great code in which, the output code artifacts will be reliable, scalable, portable, robust and maintainable. Exactly, that what is intended to be achieved through the adoption of coding standards.

3.2 The JPL Coding Standard

Since the software community realizes the importance of adopting coding standards during the software development process, many coding standards have been proposed and used during the software development. Some of these coding standards are general and applicable for several programming languages, while others are dedicated for specific language. Furthermore, some standards are well known and widely used by the software community like sun java coding standard 1999 [9] presented by Sun Micro-Systems (the first owner of Java Language), while others are self-imposed and developed by special software production companies. Some standards are targeted towards several software quality attributes, while others are targeted at certain quality attribute. Among the proposed and published coding standards, this research selected the JPL (Java Programming Language) coding standard [28] due to many reasons: (1) The primary purpose of JPL standard is reducing faults which is the addressed quality attribute by this study. (2) It is one of the most recent published standards. (3) It is published by a very reliable and reputable institution. (4) It can be covered by the available static analyzers. (5) It is dedicated for java programming language which is the underlying programming language of this study.

JPL coding standard comprises a set of 53 rules expressing bad programming practices and bugs patterns that mostly have to be avoided during writing code lists. These rules are categorized into 11 categories reflecting the usage of java language constructs. It is worth here to mention that the developers of this standard do not prioritize the rules. Furthermore, they recommend using these rules as guidelines and

they mentioned that some rules have exceptions and should not be followed to the extreme.

Although there has been developed a dedicated rule checker called semmle static analyzer [29] which implements the rules of JPL standard, this research experiments used Findbugs, PMD and Checkstyle rules checkers due to the following reasons: (1) those static analyzers are well known and widely used by java community. (2) Those static analyzers are recommended by the authors of JPL standard as alternatives for semmle static analyzer. (3) The semmle static analyzer is commercial tool.

JPL standard's rules are presented in Table 3.2 with their inspection possibility by the static analyzers used in this study. Since the aim is to empirically study the relationship between coding standard's rules violations and faults at the granular level of classes, this study ignores the JPL standard's rules that are targeted towards higher levels such as packages or systems as a whole. Such ignored rules are marked with a strike symbol in Table 3.2. Some other rules are ignored due to the lack of support for such rules by the used static analyzers. Those rules are marked with double strikes in Table 3.2. This means that among the 53 rules of the underlying standard, 43 rules are checked which means that almost 82% as percentage coverage of the JPL standard. Table F.1 in appendix F presents the JPL standard's rules and their mappings to the static analyzers' rules or checks.

Table 3.1: JPL standard's rules with their inspection possibility by the static analyzers

JPL Category	JPL Rule	PMD	CheckStyle	FindBugs
Process	"R01: compile with checks turned on." *			
	"R02: apply static analysis." *			
	"R03: document public elements."			
	"R04: write unit tests." *			
Names	"R05: use the standard naming conventions."	√	√	√
	"R06: do not override field or class names."	√	√	
Packages, Classes and Interfaces	"R07: make imports explicit."	√	√	
	"R08: do not have cyclic package and class dependencies." *			
	"R09: obey the contract for equals()."		√	√
	"R10: define both equals() and hashCode()."	√	√	√
	"R11: define equals when adding fields."			√
	"R12: define equals with parameter type Object."		√	√
	"R13: do not use finalizers."	√	√	
	"R14: do not implement the Cloneable interface."	√	√	
	"R15: do not call nonfinal methods in constructors."	√		√
	"R16: select composition over inheritance." **			
Fields	"R17: make fields private."	√		
	"R18: do not use static mutable fields."	√		√
	"R19: declare immutable fields final."	√		
	"R20: initialize fields before use."	√		
Methods	"R21: use assertions."			√
	"R22: use annotations."	√		√
	"R23: restrict method overloading."**			
	"R24: do not assign to parameters."	√	√	√
	"R25: do not return null arrays or collections."	√		√
	"R26: do not call System.exit."	√		√
Declarations and Statements	"R27: have one concept per line."	√	√	
	"R28: use braces in control structures."	√	√	
	"R29: do not have empty blocks."	√	√	√
	"R30: use breaks in switch statements."	√	√	√
	"R31: end switch statements with default."	√	√	√
	"R32: terminate if-else-if with else." **			
Expressions	"R33: restrict side effects in expressions."	√		
	"R34: use named constants for non-trivial literals."	√	√	
	"R35: make operator precedence explicit."			√
	"R36: do not use reference equality."	√	√	√
	"R37: use only short-circuits logic operators."			√

	“R38: do not use octal values.”	√		
	“R39: do not use floating point equality.	√		√
	“R40: use one result type in conditional expressions.”		√	
	“R41: do not use string concatenation operator in loops.”			√
exceptions	“R42: do not drop exceptions.”			√
	“R43: do not abruptly exit a finally block.”	√		
Types	“R44: use generics.”			√
	“R45: use interfaces as types when available.”	√	√	
	“R46: use primitive types.”			√
	“R47: do not remove literals from collections.” **			
	“R48: restrict numeric conversions.”	√		√
Concurrency	“R49: program against data races.”			√
	“R50: program against deadlocks.”			√
	“R51: do not rely on the scheduler for synchronization.” **			
	“R52: wait and notify safely.”	√		√
complexity	“R53: reduce code complexity.”	√	√	

3.3 Enforcing Coding Standards

Coding standard represents a useful tool in the fight of keeping bugs out of software products as mentioned in Section 3.1. Unfortunately, Li and Prasad [30] reported that although developers understand the importance of using coding standards, they did not adhere to such standards when the software products required to be delivered quickly. As software project manager, to ensure that your adopted coding standard is adhered to the extreme and hence acquire its benefits, you should find automated tools to enforce as many coding rules as possible from your standard. Furthermore, you should make such tools part of the developer’s integrated development environment. That way you can restrict the acceptance to those code modules that has passed the automated checking [31].

One of the easiest ways to increase the adherence to the adopted coding standard is by applying and configuring static analysis tools that have been developed to enforce

these standards (for example, PMD, FindBugs, Jlint, QAC, SQMlint and so many others). Perhaps the best option is to use a static analysis tool that includes built-in support for your chosen coding standard and/or the ability to be customized to proprietary rules of other widely used coding standards[31]. Sometimes applying one static analysis tool is not sufficient to cover all standard's rules. So, it may be necessary for software projects managers to decide to use two or more static analysis tools to fully comprehend the selected standard's rules. With these tools, indeed, software projects managers ensure that the produced code will respect selected standard and developers can better understand the code written by others. Additionally, this research work helps developers to configure the static analysis tools based on the rules/categories that have impact on software faults.

3.4 Static Analysis

Static code analysis or code review is a systematic examination of source code that can be achieved using automated tools. Static code analysis refers to the inspection of software source code without executing any piece of the target software code. In contrast to compilers which only check the syntax and type correctness of a program and accept all programs that are syntactically and type correct, static analyzers constrain that space even more to report and fix common mistakes, misinterpretations of semantics, bugs patterns and coding standards rules violations overlooked by both compilers and developers during the initial code writing phase [28]. The reported errors and warnings by such static analyzers should never be ignored, filtered or masked out to ensure the improvement of both overall quality of the software product and the developer's skills.

Static analysis is useful for many software aspects such as reasoning about runtime properties of the program code especially those properties which cause abnormal termination or unexpected results of the program. Additionally, static analysis is useful in detecting the premature abortion of the program due to for example unexpected runtime errors. Furthermore, static analysis can be used to detect runtime problems such as arithmetic overflow, division by zero, buffers overflow and array indices out of bounds without code execution. Finally static analysis is useful in test case generation, software metrics, intrusion detection, impact analysis and coding standards' rules violations[10]. This research focuses on using static analysis to detect and report coding standards' rules violations.

3.5 Static Analyzers

Among the CASE tools, there have been developed in the literature many static analyzers that are intended to be used for inspecting code according to some pre-defined standard's rules and guiding developers to locate difficult and potentially problematic areas in the source code. Those static analyzers perform such inspection using different techniques such as syntactic pattern matching, data flow analyses, type systems, model checking and theorem proving. Static analyzers also have different capabilities, some of them are intended to check specific programming language code lists, while others can check code lists from several programming languages. Since the target coding standard for this research is developed specifically for Java programming language and the target open source systems are pure Java applications, three pure Java static analyzers are used. Those static analyzers are FindBugs 2.0.3, PMD 5.0.2 and Checkstyle 5.6.1. Those

analyzers are chosen according to the recommendations in [28]. Table 3.1 shows a simple comparison for the main characteristics of those selected static analyzers.

Table 3.2: Comparison between the static analyzers used by this study

Name	Version	# of rules	Input	Interface	Technology
FindBugs	2.0.3 (2013)	417	Byte Code	CL, GUI, IDE, ANT	Syntax, dataflow
PMD	5.0.2 (2013)	295	Source Code	CL, GUI, IDE, ANT	Syntax
Checkstyle	5.6.1 (2013)	139	Source Code	CL, GUI, IDE, ANT	Syntax

3.5.1 FindBugs 2.0.3

FindBugs [32] is a bug pattern detector for Java. FindBugs employs several adhoc techniques to balance different features like precision, efficiency and usability. One of those techniques is source code matching to known problematic programming patterns. Additionally, FindBugs uses dataflow analysis for checking bugs patterns such as null pointer dereferences. Further interesting feature of FindBugs is its extendibility by writing additional custom bugs detectors using java programming language [33]. FindBugs is set to report medium priority warnings, which is the recommended setting by its documentation cite, with the bugs detectors recommended by the authors of JPL standard as shown in Table F.1 in appendix F. FindBugs can be used in three different ways: as a command line, an Eclipse plugin or an Ant target element. Since Eclipse IDE is used during this research experimentation, the analysis and report are focused on the tool being used from the Eclipse plugin. As an Eclipse plugin, the plugin comes with a FindBugs perspective as shown in figure F.1 in appendix F. For more detailed description of FindBugs' rules or bugs detectors, see [32].

3.5.2 PMD 5.0.2

PMD [34], like FindBugs, performs parsing for the Java source code into an abstract syntax tree, but it does not have a dataflow component. PMD can detect the known problematic code, stylistic guidelines whose violations can lead to suspicious bugs under some circumstances. PMD has several bugs' detectors which depend on code style practices. Those detectors are configurable and can be enabled or disabled according to the need of the developers or managers of the software projects being developed [33].In this research experiments, PMD is run with the rules recommended by the JPL coding Standard in addition to some rules that are found to be equivalents or correspondents to the rules of the underlying standard as shown in Table F.1 in appendix F. Like Findbugs, PMD can be used also in three different ways: as a command line, an Eclipse plugin or an Ant target element. The analysis and report are focused on the tool being used from the Eclipse plugin. As an Eclipse plugin, the plugin comes with a PMD perspective as shown in figure F.2 in appendix F. For more detailed description of PMD's rules or bugs detectors, see [34].

3.5.3 Checkstyle 5.6.1

Checkstyle [35] is a static analyzer to test that program code lists adhere to a set of pre-defined rules such as code layout rules, naming conventions, Javadoc documentation, and coding errors for Java programs. In this research experimentation, only the checks that correspond to JPL standard's rules are turned on as depicted in Table F.1 in appendix F. Like FindBugs and PMD, Checkstyle can be configured and made to support many coding standards. For example, a configuration file is often supplied within its installation to support the Sun Code Conventions. Other configuration files can be

supplied for other self-imposed or well-known coding standards as well. Checkstyle is available as a plug-in for many popular IDEs including Eclipse and it can be used in three different ways: as a command line, an Eclipse plugin or an Ant target element. Since Eclipse IDE is used during this research experimentation, the analysis and report are focused on the tool being used from the Eclipse plugin. As an Eclipse plugin, the plugin comes with a Checkstyle perspective as shown in figure F.3 in appendix F. For more detailed description of Checkstyle's rules, see [35].

CHAPTER 4

THE CODING STANDARD'S VIOLATIONS-BASED METRICS

This chapter explains the software metrics investigated by this study, namely, coding standard's violations-based metrics and CK metrics. Section 4.1 describes the coding standard's violations-based metrics proposed in the literature in addition to the coding standard's violations-based metrics proposed by this research work. Their definitions are presented in 4.1.1. CK metrics are described in Details in Section 4.2.

4.1 The Coding Standard's Violations-Based Metrics

Coding standards violations-based metrics are suite of metrics computed using the data collected from the software source code artifacts by means of some tools called static analyzers. Among the functionalities provided by such tools is coding rules violations detection. Those tools inspect the source code looking for the violations of coding standard's rules.

The coding standard's violations-based metrics can be defined at the standard's level, rule's level or at the category's level. These metrics can also be gathered at different granularity levels such as line's level, method's level, class's level, package's level or system's level. In this research, we defined and gathered these metrics at the class level. Reviewing the research works that have been done in the literature, it was

found that almost all previous research works used metrics based on the total number of violations and violations density. Those metrics used in the literature suffer from many limitations such as, the lack of distinguishing between violations diversity at the standard level, the lack of distinguishing between violations diversity at the category level, the lack of distinguishing between categories of violations and the lack of distinguishing between violations severity.

The results of the static analyzers' inspection are violations reports for the coding rules whose equivalent or correspondent tools' rules are turned on. The violations report contains information about the coding rule's being violated in the inspected module such as the module name, the violated rule, the code line number in which the rule is violated. The violations report for each class is inserted into the violations database using the tool presented in section 5.3. At this point, the metrics values can be calculated and retrieved from the database by means of SQL queries. As mentioned in section 3.5, the proposed metrics are derived according to the coding rules' categorization presented and adopted by the JPL coding standard. In the following section, each one of the proposed metrics is defined and presented with an example.

4.1.1 Definitions of Coding Standard's Violations-Based Metrics

In this section, each one of the proposed metrics is defined and presented with its scale values. In the next sections, class code size means the LOC excluding the blank and comments lines.

- **M1: The Percentage of Standard's Rules being Violated per class (PSRV)**

  PSRV of a class C is the percentage of the JPL coding standard's rules that have been violated in C. It ranges from 0% to 100%.

M2: The Percentage of Standard's Rules being Violated normalized by the class code size (PSRVD)

PSRVD of a class C is the percentage of JPL coding standard's rules that have been violated in C which in turn, is normalized by the class code size. Since this metric depends on the class code size, it is difficult to specify its range borders values.

- **M3: The Percentage of Category's rules being violated in the class**

As discussed in the previous sections, the metrics are defined according to the coding rules' categories proposed by the JPL coding standard. So, this metric can be divided into 10 sub-metrics based on the coding rules' categories adopted by the JPL standard as follows:

- **M3.1: The Percentage of Names Category's Rules being violated in the class (PNCRV)**

PNCRV of a class C is the percentage of names category's rules that have been violated in C. Since names category has only 2 rules, this metric takes 0%, 50% or 100%.

- **M3.2: The Percentage of Packages, Classes and Interfaces Category's Rules being Violated in the class (PPCICRV)**

PPCICRV of a class C is the percentage of Packages, Classes and Interfaces category's rules that have been violated in C. PPCICRV ranges from 0% to 100%.

- **M3.3: The Percentage of Fields Category's Rules being Violated in the class (PFCRV)**

  PFCRV of a class C is the percentage of Fields category's rules that have been violated in C. Since Fields category has only 4 rules, this metric takes one of the following values: 0%, 25%, 50% or 100%.

- **M3.4: The Percentage of Methods Category's Rules being Violated in the class (PMCRV)**

  PMCRV of a class C is the percentage of Methods category's rules that have been violated in C. Since Methods category has only 5 rules, this metric takes one of the following values: 0%, 20%, 40%, 60%, 80% or 100%.

- **M3.5: The Percentage of Declarations and Statements Category's Rules being Violated in the class (PDSCRV)**

  PDSCRV of a class C is the percentage of declarations and statements category's rules that have been violated in C. Since declarations and statements category has only 5 rules, this metric takes one of the following values: 0%, 20%, 40%, 60%, 80% or 100%.

- **M3.6: The Percentage of Expressions Category's Rules being Violated in the class (PExpCRV)**

  PExpCRV of a class C is the percentage of expressions category's rules that have been violated in C. It ranges from 0% to 100%.

- **M3.7: The Percentage of Exceptions Category's Rules being Violated in the class (PExcCRV)**

  PExcCRV of a class C is the percentage of exceptions category's rules that have been violated in C. Since exceptions category has only 2 rules, this metric takes 0%, 50% or 100%.

- **M3.8: The Percentage of Types Category's Rules being Violated in the class (PTCRV)**

  PTCRV of a class C is the percentage of types category's rules that have been violated in C. Since types category has only 4 rules, this metric takes one of the following values: 0%, 25%, 50%, 75% or 100%.

- **M3.9: The Percentage of Concurrency Category's Rules being Violated in the class (PConCRV)**

  PConCRV of a class C is the percentage of Concurrency category's rules that have been violated in C. Since Concurrency category has only 3 rules, this metric takes one of the following values: 0%, 33.33%, 66.66% or 100%.

- **M3.10: The Percentage of Complexity Category's Rules being Violated in the class (PComCRV)**

  PComCRV of a class C is the percentage of Complexity category's rules that have been violated in C. Since Complexity category has only 1 rule, this metric takes either 0% or 100%.

- **M4: The Percentage of Category's rules being violated in the class, normalized by the class code size**

  As discussed in the previous sections, the metrics are defined according to the coding rules' categories proposed by the JPL coding standard. So, also this metric can be divided into 10 sub-metrics based on the coding rules' categories adopted by the JPL standard as follows:

- **M4.1: The Percentage of Names Category's Rules being Violated in the class normalized by the class code size (PNCRVD)**

  PNCRVD of a class C is the percentage of names category's rules that have been violated in C which in turn, is normalized by the class code size. Since this metric depends on the class code size, it is difficult to specify its range borders values.

- **M4.2: The Percentage of Packages, Classes and Interfaces Category's Rules being Violated in the class normalized by the class code size (PPCICRVD)**

  PPCICRVD of a class C is the percentage of Packages, Classes and Interfaces category's rules that have been violated in C which in turn is normalized by the class code size. Since this metric depends on the class code size, it is difficult to specify its range borders values.

- **M4.3: The Percentage of Fields Category's Rules being Violated in the class normalized by the class code size (PFCRVD)**

  PFCRVD of a class C is the percentage of Fields category's rules that have been violated in C which in turn, is normalized by the class code size. Since this metric depends on the class code size, it is difficult to specify its range borders values.

- **M4.4: The Percentage of Methods Category's Rules being Violated in the class normalized by the class code size (PMCRVD)**

  PMCRVD of a class C is the percentage of Methods category's rules that have been violated in C which in turn, is normalized by the class code size. Since this metric depends on the class code size, it is difficult to specify its range borders values.

- **M4.5: The Percentage of Declarations and Statements Category's Rules being Violated in the class normalized by the class code size (PDSCRVD)**

  PDSCRVD of a class C is the percentage of Declarations and Statements category's rules that have been violated in C which in turn, is normalized by the class code size. Since this metric depends on the class code size, it is difficult to specify its range borders values.

- **M4.6: The Percentage of Expressions Category's Rules being Violated in the class normalized by the class code size (PExpCRVD)**

  PExpCRVD of a class C is the percentage of expressions category's rules that have been violated in C which in turn, is normalized by the class code size. Since this metric depends on the class code size, it is difficult to specify its range borders values.

- **M4.7: The Percentage of Exceptions Category's Rules being Violated in the class normalized by the class code size (PExcCRVD)**

  PExcCRVD of a class C is the percentage of exceptions category's rules that have been violated in C which in turn, is normalized by the class code size. Since this

metric depends on the class code size, it is difficult to specify its range borders values.

- **M4.8: The Percentage of Types Category's Rules being Violated in the class normalized by the class code size (PTCRVD)**

  PTCRVD of a class C is the percentage of Types category's rules that have been violated in C which in turn, is normalized by the class code size. Since this metric depends on the class code size, it is difficult to specify its range borders values.

- **M4.9: The Percentage of Concurrency Category's Rules being Violated in the class normalized by the class code size (PConCRVD)**

  PConCRVD of a class C is the percentage of Concurrency category's rules that have been violated in C which in turn, is normalized by the class code size. Since this metric depends on the class code size, it is difficult to specify its range borders values.

- **M4.10: The Percentage of Complexity Category's Rules being Violated in the class normalized by the class code size (PComCRVD)**

  PComCRVD of a class C is the percentage of Complexity category's rules that have been violated in C which in turn, is normalized by the class code size. Since this metric depends on the class code size, it is difficult to specify its range borders values.

- **M5: The Percentage of Standard's Categories being Violated in the class (PSCV)**

  PSCV of a class C is the percentage of JPL Standard's Categories that have been violated in C. It ranges from 0% to 100%.

- **M6: The Percentage of Standard's Categories being Violated in the class normalized by the class code size (PSCVD)**

  PSCVD of a class C is the percentage of JPL Standard's Categories that have been violated in C which in turn, is normalized by the class code size. Since this metric depends on the class code size, it is difficult to specify its range borders values.

4.2 CK Metrics

This section describes CK metrics in details. CK metrics are well-known suite of product metrics for measuring the structural properties of OO systems' classes.

Structural class's properties using CK metrics have been studied in literature extensively by many research studies with association to different quality attributes. Among those addressed quality attributes are the fault proneness and fault density of classes. Therefore, in this research we use CK metrics [17] in two-folds. First, they are used to build prediction models for the sake of comparing their prediction power with the prediction power of our proposed metrics' models. Second, in order to achieve comprehensive prediction model which considers both suites of metrics as presented in chapter 7.

4.2.1 CK Metrics Definitions

This section presents formal definitions for the CK metrics:

- **Weighted methods per class (WMC):**

  This metric measures the static complexity of individual classes. With the assumption that all methods of a class are equally complex, then WMC is the number of local methods.

- **Depth of inheritance (DIT):**

  It measures the position of the class in the inheritance hierarchy. It is defined as the length of the longest path from the node to the root of inheritance tree.

- **Number of children (NOC):**

  It measures the number of classes that inherit directly from a class.

- **Coupling between objects (CBO):**

  This metric counts the number of other classes that are coupled to a class either as a client or a supplier. A class is coupled to another if it uses the member functions and/or instance variables of the other class.

- **Response for a class (RFC):**

  This metric measures the cardinality of the response set of the class. The response set of the class is a set of methods that can potentially be executed in response to a message received by an object of that class.

  RFC = number of local methods + number of methods called by local methods.

- **Lack of cohesion in methods (LCOM):**

  LCOM is a measure of not connected method pairs in a class. LCOM = 100% - the average cohesion for class data members. Where, a method is cohesive when it performs a single task.

CHAPTER 5

CODING RULES VIOLATIONS DATABASE DESIGN

Since the primary goal of this research study is to investigate the relationship between the coding standards' rules violations and the existence of the actual software faults, it is necessary to have in hand both software faults data and coding standards' rules violations data.

Holding or storing such data in a suitable form so that it can be manipulated and used in analysis easily and effectively is a vital step towards achieving study's goal. XML and relational data base forms are popular forms and used widely in almost every application where the data warehousing is necessary. So in this research, the relational form is used due to its simplicity and efficiency.

5.1 Relational Database Model

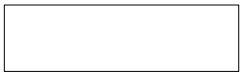
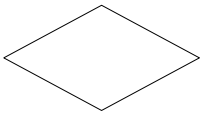
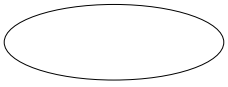
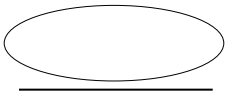
Relational Data Base, in a simple definition is a shared repository of data [36]. The relational model depicts the database as a set of relations. Each one of those relations looks like a table of values or like a flat file of records. Once a relation is considered as a flat file of records, each record in the table represents a collection of related data values [37].In the formal relational model terminology, a row is called a *tuple,* a column header is called an *attribute,* and the table is called a *relation.* The data type describing the types of values that can appear in each column is called a *domain*. For more information, see [37]. Coding rules violations database is relatively small in terms of the number of tables

and relations. It consists of only 8 tables or entities. Figure 5.1 presents the database Tables (entities) and the relations among them while figure 5.2 presents the database tables and their columns (attributes) in addition to the integrity constraints applied on them.

5.2 Entity Relationships Model

ER model is a popular high-level conceptual data model. This model and its variations are frequently used for the conceptual design of database applications. The ER model describes data as entities, relationships, and attributes using specific notations as shown in Table 5.1. The ER model uses the rectangle to represent the entity, diamond to represent the relation and ellipse to represent the attribute.

Table 5.1: The entity relationships model's notations

Symbol	Description
	Used to represent the entity which is a table in the database.
	Used to represent the relationship between entities.
	Used to represent the attribute of an entity.
	Used to represent the attribute of the entity which is specified as primary key

The entity relationships model for coding rules' violations database is shown in figure 5.1. As presented in this figure, all entities' relationships are binary (between two entities). For example, one to many relationship between the entity systems and the entity System_clasees which means that many classes belong to one system. One to one relationship between the entity System_classes and the entity Tested_source which means that each system's class inspected only one time during our experiment. One to many relationship between the entity Tested_source and the entity Source_violations which means that the system's class may violates several coding rules. One to many relationship between the entity Categorization_type and the entity Analyzers_categories which means that the categories in Analyzers_categories categorized according to the type specified in Categorization_type. One to many relationship between analyzers_categories and analyzers_cat_rules which means that many rules belong to one category. One to many relationship between the entity JPL_categories_rules and the entity Analyzers_cat_rules which means that many rules from Analyzers_cat_rules equivalent to one rules from the JPL_categories_rules. One to many relationship between the entity Analyzers_cat_rules and the entity Source_violations which means that one rule from Analyzers_cat_rules may be violated several times in Source_violations.

The detailed ER Models in which the entities with their attributes in addition to the relations between those entities are presented in appendix G.

Figure 5.1: The entity relationship model for the coding rules violations database

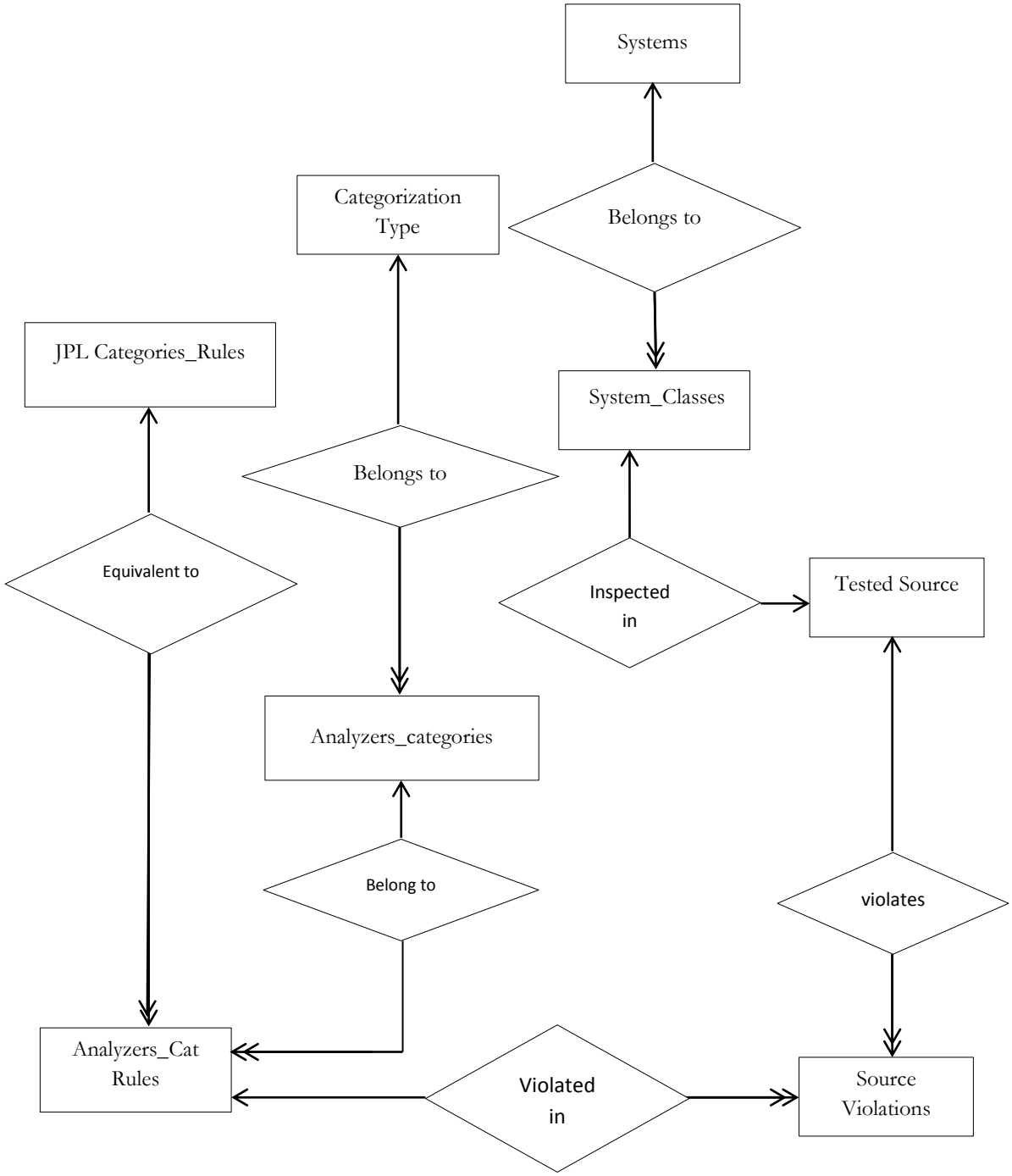


Figure 5.2 shows the underlying schema of the coding standard's rules violations database designed for this research study. As presented in this figure, Table categorization_type holds the coding rules categorization types. This table is added to the database schema to give more flexibility in adopting several types of rules' categorization like Sun categorization or even the rules' categorization adopted by the static analyzers themselves for example to evaluate the relationships between the static analyzers warnings and different software quality attributes. Table categorization_type is the parent for table analyzers_categories which in fact hold the information about all coding rules' categorization adopted by the three static analyzers(Checkstyle, FindBugs and PMD) used by this study. Table analyzers_categories is the parent table for table analyzers_cat_rules which hold the information about the coding rules of each tool's category in addition to the mapping information of tools' rules into JPL coding standard's rules.

Figure 5.2: The coding rules violations database schema

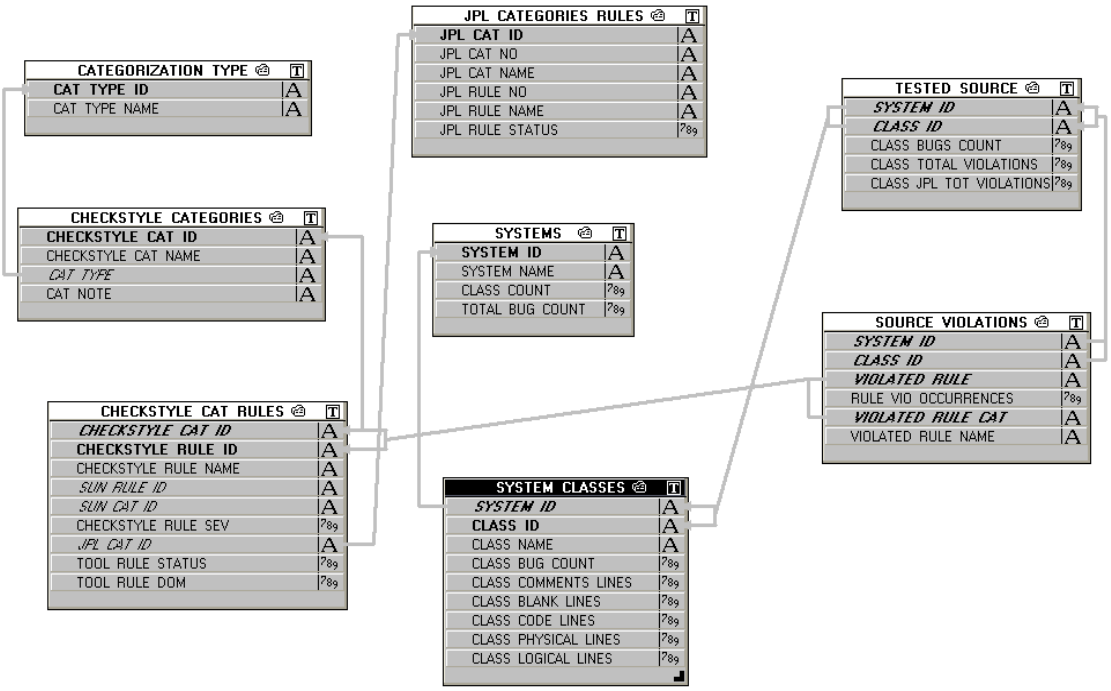


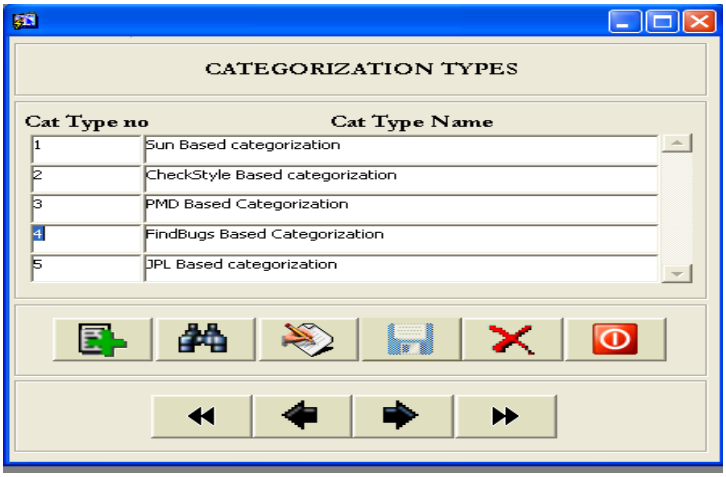
Table JPL_categories_rules hold the information about the JPL coding standard's rules and categories which is the chosen standard for our study. Tables systems and system_classes hold the information about the chosen open source systems for this study and their classes. Tables tested_source and source_violations hold the coding standard's rules' violations data reported by the three static analyzers used to inspect the open source systems' classes.

5.3 Graphical User Interfaces

A graphical interface (GUI) typically displays a schema to the user in diagrammatic form in which the user can query and/or manipulate with the database easily and efficiently. This section presents the designed tool's GUIs through which the database is initialized for the sake of receiving and storing coding standard's rules violations reported by the static analyzers.

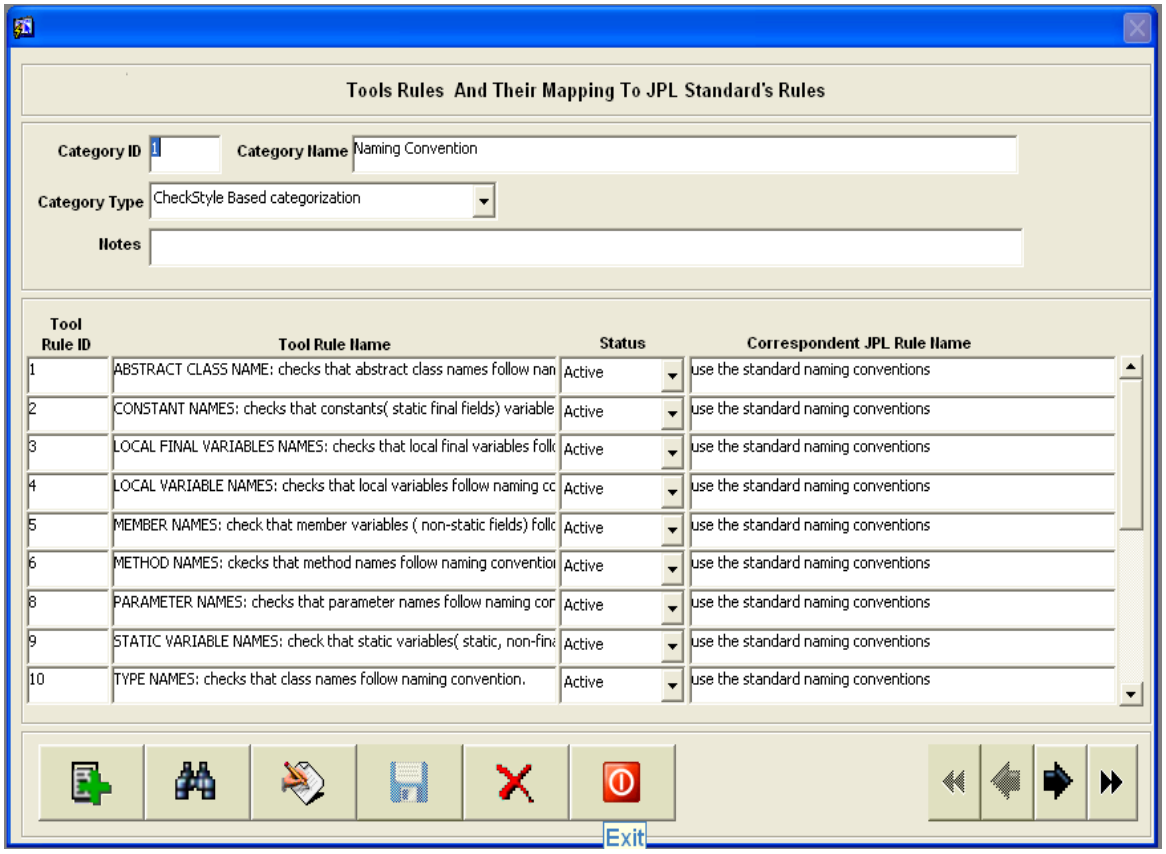
Categorization Types: through this window, the types of coding rules categorization are inserted into the database such as Sun Microsystems coding rules categorization or any other coding standard rules categorization. This window provides the researcher by the required flexibility to choose the suitable coding rules categorization. Figure 5.3 shows the categorization types window.

Figure 5.3: The categorization types window



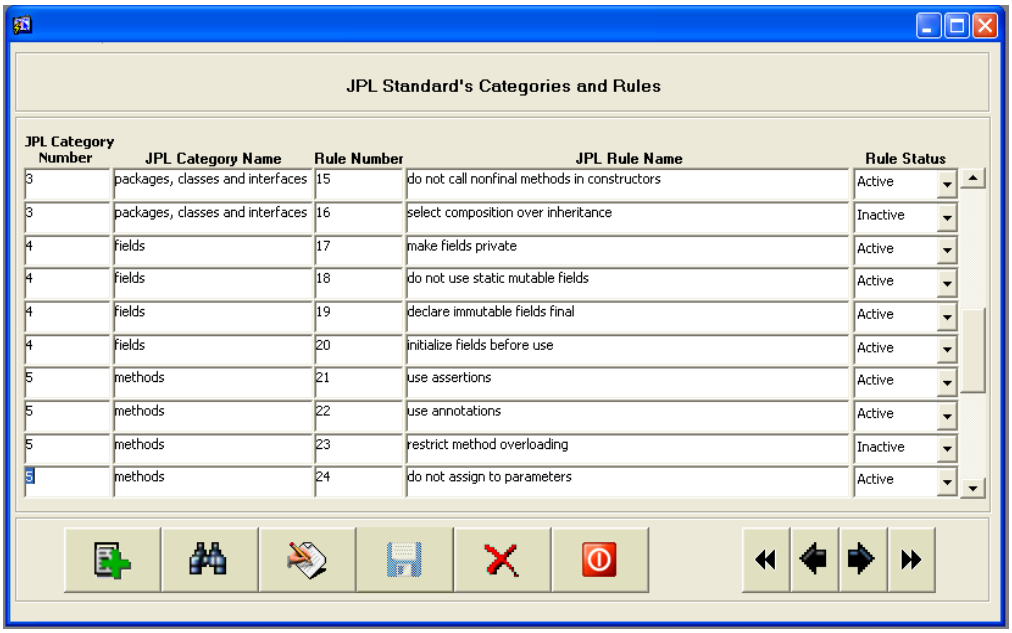
Tools' rules and their mappings into JPL standard's rules: this window is shown in figure 5.4 and used to insert the static analyzers' rules and their mapping to the JPL standard's rules. The source code is inspected against the rules inserted using this window.

Figure 5.4: The tools' rules and their mapping to JPL standard's rules window



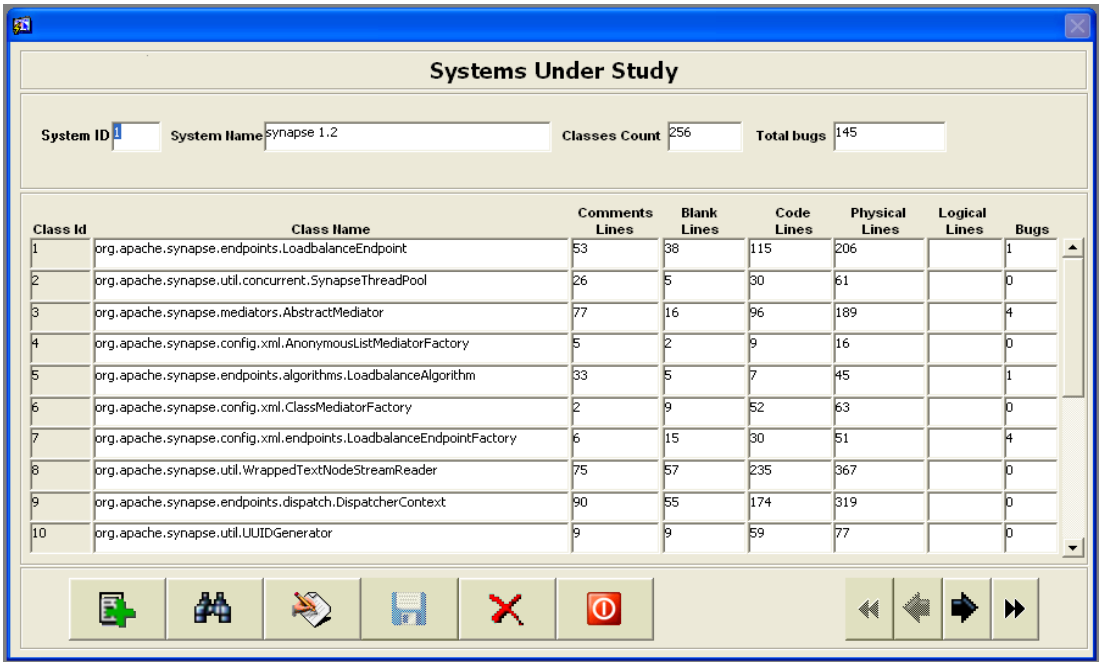
JPL standard categories and rules: this window is shown in figure 5.5 and used to insert the categories and their rules for the chosen coding standard for the study. The chosen coding standard in this research case is the JPL coding standard.

Figure 5.5: The JPL standard's categories and rules window



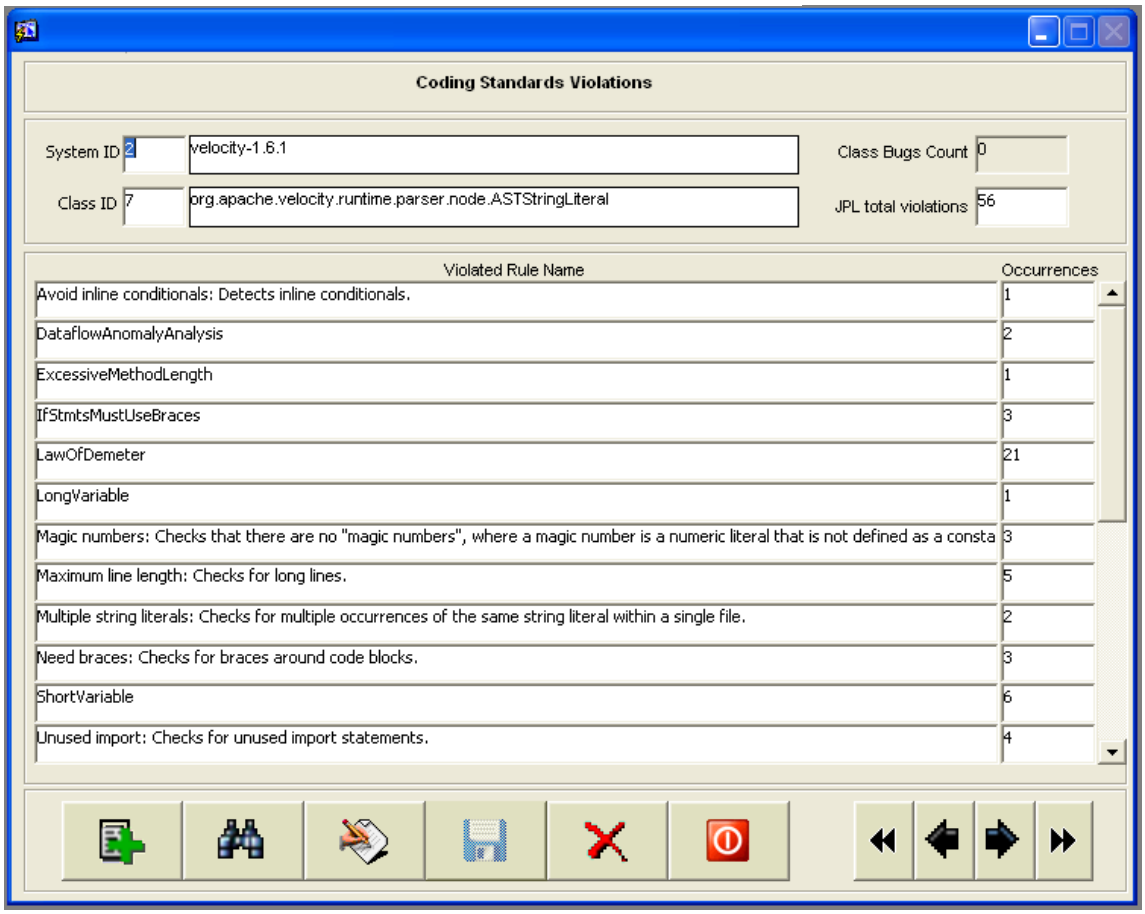
Systems under study: this window is shown in figure 5.6 and used to insert the required data about the open source projects investigated by this study. The required data about such open source projects is classes' names, sizes and the number of bugs in each class.

Figure 5.6: The systems under study window



Coding standard violations: this window is shown in figure 5.7 and used to insert the violations data reported by the static analyzers tools used to inspect the classes of the open source projects investigated by this study.

Figure 5.7: The coding standard violations window.



CHAPTER 6

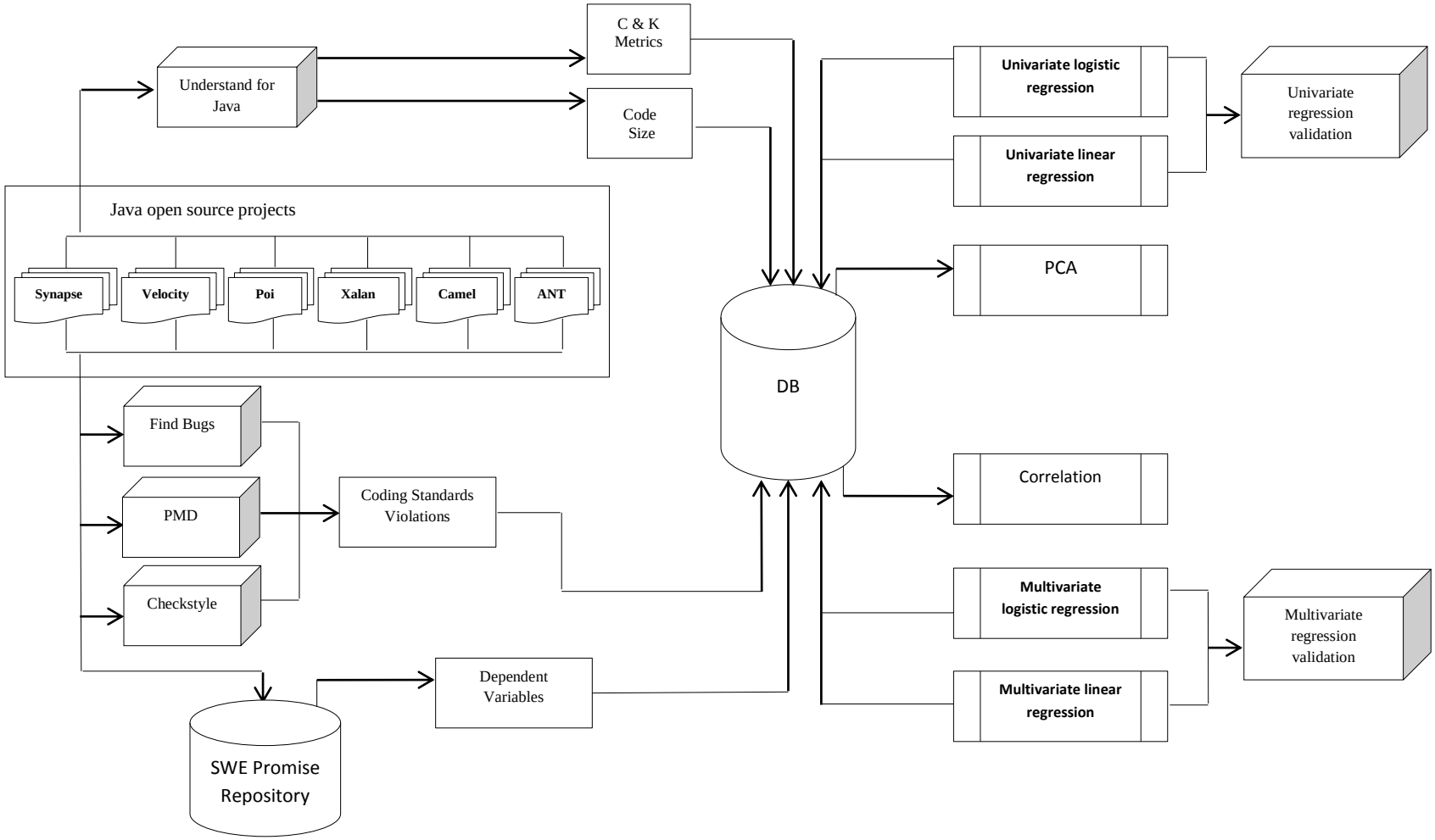
EMPIRICAL STUDY DESIGN

Software quality can be defined in terms of different quality attributes, such as maintainability, reliability, portability, flexibility, changeability and functional correctness. Several metrics have been proposed in the literature to capture such software quality aspects. Those software metrics can be elicited from different artifacts of the software system. But how do we know which metrics are useful in capturing important quality aspects?

Empirical validation of software metrics is essential to ensure their practicality and usefulness. To obtain empirical evidence and to answer the research questions, provided in Section 1.2, a large set of metrics called coding standards violations-based metrics are collected and analyzed. The values of these metrics are collected from six open source software systems, Ant-1.7.0, Apache-Camel-1.6.0, Poi-3.0, Synapse-1.2, Velocity-1.6.1 and Xalan-2.6.0. This chapter describes the procedure to empirically validate this set of metrics as software faults predictors. Figure 6.1 summarizes the empirical procedure, which will be detailed in the rest of this chapter.

This chapter is organized as follows. A detailed description about data collection is described in Section 6.1. This is followed by the research hypotheses in Section 6.2. Then, in Section 6.3, the empirical data analysis methodology is described.

Figure 6.1: The empirical validation framework



6.1 Empirical Data Collection

The coding standards violations-based metrics are collected from six open source software systems, Ant-1.7.0, Apache-Camel-1.6.0, Poi-3.0, Synapse-1.2, Velocity-1.6.1 and Xalan-2.6.0. All systems are long-lived, of reasonable size in terms of the number of classes, and from different application domains. Working on long-lived systems prevents results from being biased by the potential data fluctuations experienced during short period of time [12]. Additionally, selecting a bigger set of systems from different domains makes the obtained findings more generalizable. Furthermore, investigating reasonable-size systems in terms of the number of classes increasing the number of data points which is considered a good feature for statistical analysis [38]. Here are some details about the open source systems investigated by this study. For example, in terms of the number of classes, Ant-1.7.0 has 746 Classes, Apache-Camel-1.6.0 has 933 classes, Poi-3.0 has 439 classes, Synapse-1.2 has 256 classes, Velocity-1.6.1 has 229 classes and Xalan-2.6.0 has 885 Classes. In addition to the number of classes in each system, number of bugs, system size in terms of Lines of Code (LOC excluding comments and blank lines) and the percentage of faulty classes in each system (each class has one bug or more is counted) are presented in Table 6.1. As shown in the table, each system has different number of classes, different number of bugs, size, and percentage of fault-prone classes.

Table 6.1 Descriptive statistics of the systems under study

System ID	System Name	Bugs count	System Code Size (LOC)	Number of Classes	Percentage of Faulty Classes%
1	Synapse-1.2	145	19554	256	86 (33.98%)
2	Velocity-1.6.1	190	25241	229	78 (34.06%)
3	Poi-3.0	500	51402	439	281 (63.43%)
4	Xalan-2.6.0	625	151485	885	411 (46.44%)
5	Camel-1.6.0	500	56444	933	188 (20.15%)
6	Ant-1.7.0	338	87741	745	166 (22.28%)

6.1.1 Data Collection

To obtain class size in terms of source lines of code (LOC) for the sake of calculating some of the proposed coding standards violations-based metrics, a reverse engineering tool called Understand 3.1 (Build 693) (Copyright © 1996-2013 Scientific Toolworks, Inc.), the analyst edition [39] is used. The tool belongs to source code analyzers that help developers to understand their software projects. The Understand tool analyzes Java source code to create an indexed repository for the relations and structural properties contained within the source code artifacts. The repository is then used to learn about the source code. The tool receives the classes of certain system of the open source software systems as an input and produces many software product metrics such as CK metrics. However, the tool is configured to only measure the CK metrics in addition to the class size in terms of source lines of code (LOC).

To calculate the coding standards violations-based metrics, three static analysis tools called FindBugs 2.0.3, PMD 5.0.2, CheckStyle 5.6.1 are used. All those tools are popular and widely used to inspect java source code. They are powerful, yet intuitive and easy to use. These tools can be used in three different ways: as a command line, an Eclipse plugin or an Ant target element with almost any operating system platform. FindBugs and PMD provide an extra feature in which users can export the violations reports into an XML or Excel files for further processing. However, to the best of our knowledge, Checkstyle lacks such feature which in turn imposes manual processing for its generated reports.

Furthermore, all of these three tools provide some sort of severity for their rules or checks. Unfortunately, some conflicts are found between the prioritization of equivalent rules of these tools. These conflictions in severity of tools' rules was the reason behind discarding rules' severity to be one of this research objectives in which the JPL standard's rules will be prioritized from the point of view of functional correctness. These tools also enable users to configure their inspection according to the adopted coding standard, bugs patterns or bad practices they looking for. More details about them are presented in the section 3.4.

Since the underlying coding standard of this study was JPL coding standard for java programming language, the experiments' settings enabled totally 176 rules from different categories of rules for each tool according to the rules mappings provided in Table F.1 in the appendix F. From the totally enabled rules, the tools' portions was 55, 73, 48 rules for FindBugs, PMD and Checkstyle respectively. Another important point that deserves to be mentioned here is, although each tool has its own categorization for its

rules, this research ignored such tools rules based categorizations and adopted the categorization provided by the JPL coding standard. Table F.1 in the appendix F provides the categorization for each JPL standard's rule as specified by both the JPL standard and the tools.

For the coding standards violations-based metrics to be collected, the analysis and report are focused on the tools being used from the Eclipse plugin. The plugin for each tool comes with its own perspective. Since both Checkstyle and PMD works only on source code (not byte code), the java open source projects are imported into the eclipse to be analyzed by Checkstyle and PMD. The generated violations reports by both are then inserted into the coding rules violations database using the developed tool for further analysis. Regarding FindBugs, instead of importing the source code form of the systems under study, the executable forms (.Jar) of the systems are imported into the Eclipse to be analyzed by FindBugs because it works only on the Byte code (not source code). The generated violations report is then inserted into the coding rules violations database for the purpose of doing further analysis. Having all generated coding standard violations data in the database, the coding standards violations-based metrics can be retrieved as SQL queries for each class of each open source project. At this point, the coding standards violations-based metrics data are then plugged into MS Excel 2010 for further analysis.

6.1.1.1 Data Used In the Prediction of Fault-Proneness

The faults data for each class of the systems under study is collected from the PROMISE software engineering repository. Additionally, the coding standard violations-based metrics data is collected to be combined with fault-proneness data (each class has

one or more faults is set to True T, otherwise is set to False F) in a CSV file format. Each class in the CSV file represents a data point or observation. At this point, it is possible to investigate the usefulness and practicality of the proposed coding standard violations-based metrics in predicting the fault-proneness of java classes.

6.1.1.2 Data Used In the Prediction of Fault Density

The faults data for each class of the systems under study is collected from the PROMISE software engineering repository. Additionally, the class code size data extracted by the Understand tool is used to calculate the faults density in each class of the target set of systems. The density data for each class is then combined with the coding standard violations-based metrics data and plugged into CSV file format. Each class in the CSV file represents a data point or observation. At this point, the investigation of the usefulness of the proposed coding standard violations-based metrics in predicting the fault density is possible.

6.1.2 Dependent and Independent Variables

As mentioned earlier, the main goal of this study is to empirically investigate and validate the practicality and usefulness of the coding standard violations-based metrics, along with CK metrics, in identifying the fault-prone classes and predicting the fault density of the target set of open source systems under study.

Thus, as dependent variable for the fault-proneness we used a binary variable called fault-proneness (FP). The actual value of this variable expressing whether the class is “faulty” or “not faulty”. FP set to 0 if the class has no faults and 1 otherwise according to the classes’ faults data. To be able to determine whether the coding standard violations-based metrics are useful predictors of the faults, a standard classification

technique, called logistic regression is performed, which is based on predicting event probabilities. In this context, the event is the fault of the considered systems classes.

This probability is described as a function of the structural properties of the classes (namely CK metrics), the coding standard violations of the class (namely coding standard violations-based metrics), or a combination of the two. These metrics, CK and the coding standard violations-based metrics, represent the independent variables. This study considered a total of 6 CK metrics and 24 coding standard violations-based metrics. A detailed description for the coding standard violations-based metrics and CK metrics is given in Chapter 4.

Regarding fault density, the fault density dependent variable is defined as the number of faults in the class which is then normalized by the size of that class in term of KLOC (KLOC excluding comments and blank lines). To determine whether the coding standard violations-based metrics are useful predictors for the fault density, a standard technique, called linear regression is used. The fault density is described as a function of the structural properties of the classes (namely CK metrics), the coding standard violations of the class (namely coding standard violations-based metrics), and a combination of the two. These metrics, CK and the coding standard violations-based metrics, represent the independent variables.

6.2 Research Hypothesis

The hypotheses for this research are as follows:

- **H1:** There is a relationship between coding standards violations and fault proneness at the class level.

- **H2:** Coding standard violations-based metrics are more accurate in predicting the class fault-proneness than are the CK metrics.
- **H3:** Prediction models based on both CK and Coding standard violations-based metrics provide better prediction accuracy, in identifying the class fault-proneness, than those based on the CK metrics.
- **H4:** Prediction models based on both CK and Coding standard violations-based metrics provide better prediction accuracy, in identifying the fault-prone classes, than those based on the Coding standard violations-based metrics.
- **H5:** There is a relationship between coding standards violations and fault density of classes.
- **H6:** Coding standard violations-based metrics are more accurate in estimating the class fault density than are the CK metrics.
- **H7:** Prediction models based on both CK and Coding standard violations-based metrics provide better prediction accuracy, in estimating the class fault density, than those based-on the CK metrics.
- **H8:** Prediction models based on both CK and Coding standard violations-based metrics provide better prediction accuracy, in estimating the class fault density, than those based on the Coding standard violations-based metrics.

6.3 Empirical Data Analysis Methodology

The methodology followed in this research for analyzing the data obtained for each metric consists of the following stages: (1) Principal Component Analysis, (2) correlation analysis, (3) attribute selection (4) building regression models, (5) and validating the regression models.

6.3.1 Data Analysis Tools

The Statistical Package for the Social Sciences (SPSS) version 21 for Microsoft Windows is selected to perform: the Principal Component Analysis, the correlation analysis, and obtaining the goodness indicators such as R^2, adjusted R^2 of the explanatory regression models and Nagelkerke R^2, Cox & Snell R^2 and -2Log likelihood of the logistic regression models. This statistical software program has been cited as one of the rigorous software packages used in literature.

For the attribute selection and building and validating the regression models, the Weka [40] is used, Weka is an open source machine-learning tool. After collecting all class-level metrics and plugging them into an Excel spreadsheet, they are feeded to WEKA as CSV file format. WEKA's GUI makes it easy to build the regression models and evaluate prediction performance for the built models.

6.3.2 Principle Component Analysis

Principal component analysis (PCA) refers to the process by which principal components are computed for the subsequent use of these components in understanding the data [41]. In other words, PCA is a standard technique to derive a small number of linear combinations (principal components) of a set of variables that retain as much of the information in the original variables as possible. If a group of variables in a data set are strongly correlated, these variables are likely to measure the same underlying dimension. The sum of the squares of the coefficients of the standardized variables in one linear combination is equal to one.

Principal-Component Method (PC method) is used to maximize the sum of squared loadings of each factor extracted in turn. The PC method aims at constructing

new variable (Pi), called Principal Component (PC) out of a given set of variables Xj' s(j = 1,2,..., k).

In order to identify these variables, and interpret the PCs, the rotated components are considered. As the dimensions are independent, orthogonal rotation is used. There are various strategies to perform such rotation. This research used the Varimax rotation, which is the most frequently used strategy in literature [42].

In the prediction models, the PCs are not used as independent variables due to the following reasons: First of all, the goal of using PCA is to interpret the results from regression analyses according to the results obtained from PCA, for example, specifying to which PCs, the metrics that are found to be significant belong to. In other words, this shows which dimensions are the main drivers of fault-proneness and fault density and may help explain why this is the case. Moreover, PCA helps to know whether the metrics from the two different suites (CK and coding standard violations-based metrics) are measuring different dimensions of functional correctness or whether they are measuring the same thing. Second, principal components are always specific to the particular data set on which they have been computed, and may not be representative for other datasets. Thus, a model built using principal components is likely to be not applicable across different systems.

6.3.3 Correlation Analysis

In order to determine which metrics can be used individually as fault predictors, two correlation techniques are applied: Spearman and Kendall’s tau techniques. Each one of these techniques is performed between each individual metric of the coding standard violations-based suite and the fault-proneness of the class. The same correlation

techniques are performed between each individual metric of the coding standard violations-based suite and the fault density of the class as well. The spearman correlation test is the most suitable test for this study since the data is not normally distributed.

Although the Spearman's rank correlation coefficient is the most commonly used correlation coefficient [16], Kendall's tau correlation is also decided to be performed to get further insight into the significance of the correlation and to confirm the results obtained by the spearman correlation analysis. Spearman's rank correlation coefficient is calculated as follows. The x and y values are placed in ascending order and ranked separately by assigning 1 to the smallest value, 2 to the next smallest, and so on, if two or more raw values are equal, each is given the average of the related rank values. Then, the values of the variables (i.e. the observations) are sorted according to the rank of the first variable. After that, correlation coefficient is calculated from the following formula:

$$r_s = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \qquad (6.1)$$

Where di is the difference between the ranking values corresponding to certain observation (x, y) in the raw data. For more information about the spearman correlation, see [38]. *P*-value is used to judge the significance of the correlation.

Similar to Spearman's rank-order correlation coefficient, Kendall's tau correlation coefficient is designed to capture the association between two ordinal (not necessarily interval) variables. Its estimate (denoted τ) can be expressed as follows:

$$\tau = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} sgn(x_i - x_j) sgn(y_i - y_{j)}}{n(n-1)} \qquad (6.2)$$

Where

$$sgn(x_i - x_j) = \begin{cases} 1 & if(x_i - x_j) > 0 \\ 0 & if(x_i - x_j) = 0 \\ -1 & if(x_i - x_j) < 0 \end{cases} \quad \text{And}$$

$$sgn(y_i - y_j) = \begin{cases} 1 & if(y_i - y_j) > 0 \\ 0 & if(y_i - y_j) = 0 \\ -1 & if(y_i - y_j) < 0 \end{cases}$$

This coefficient quantifies the discrepancy between the number of concordant and discordant pairs. Any two pairs of ranks (xi, yi) and (xj, yj) are said to be concordant when xi < xj and yi < yj, or when xi > xj and yi > yj, or when (xi - xj)(yi - yj) > 0. Correspondingly, any two pairs of ranks (xi, yi) and (xj, yj) are said to be discordant when xi < xj and yi > yj, or when xi > xj and yi < yj, or when (xi - xj)(yi - yj) < 0. "Similar to the two previous correlation coefficients, Kendall's tau ranges from -1 to +1, with the absolute value of τ indicating the strength of the monotonic relationship between the two variables. However, Kendall's tau can be 1 for even a wider range of scenarios than Spearman's correlation coefficient" [43].

6.3.4 Attribute Selection

Variable selection is a preliminary step which applied in multivariate data analysis especially when having a large number of independent variables. In such case, there is a possibility that some of these variables contain redundant or noisy information. Furthermore, there can be a high correlation between independent variables which in turn can negatively affect the regression results as it can lead to unstable coefficients (large standard error and low t-values). In these cases, a variable or feature selection procedure is required to remove the collinearity among variables and in order not to fall into the over-fitting problem.

"Overfitting is a phenomenon in which a predictive model learns the idiosyncrasy of the data; then, the noise is modeled as well, and the model loses its generalization feature." [44]. Thus, it is useful to be able to reduce the model to contain only those variables which provide important information about the dependent variable. These variables are then used as the basis for further modeling steps explained below.

In this research, the Weka tool is used for attribute selection with the default setting as follows. The program uses the CfsSubsetEval evaluator as a default attribute evaluator. This evaluator considers the predictive value of each attribute individually, along with the degree of redundancy among them. The locallyPredictive property of the evaluator (set by default to true) allows adding the attributes with the highest correlation with the dependent variable as long as there is no already an attribute in the subset that has a higher correlation with the attribute in question [45].

Regarding the research method for selecting the attributes, the default setting is the BestFirst search method. It searches the space of attribute subsets by greedy hill climbing augmented with a backtracking facility.

6.3.5 Prediction Models

This research applied an empirical study to investigate the practicality and usefulness of using coding standard violations-based metrics as predictors for functional correctness in terms of fault proneness and fault density. Accordingly, we performed multivariate and univariate logistic and linear regression techniques. The regression model is considered univariate if it is only features one independent variable and multivariate otherwise. In this case study, the power of the 24 proposed coding standard violations metrics is investigated in predicting both fault proneness and fault density of

Java classes. Univariate regression technique is applied to study the prediction power of each metric separately while multivariate regression technique is performed to study the prediction power of the whole set of coding standard violations metrics in addition to the combination of coding standard violations metrics with CK metrics as discussed in the next sections.

6.3.5.1 Building Univariate Prediction Models

Since this research problem is of two types, classification and regression, two types of univariate regression models are built. Univariate logistic regression models and univariate linear regression models. These models are built to predict the fault-proneness and the fault density of the classes, respectively.

6.3.5.1.1 Univariate Logistic Regression Models

Univariate logistic regression is useful for situations in which you want to be able to predict the presence or absence of a characteristic or outcome based on values of one predictor variable [38, 46, 47]. This means that those types of models are useful and suitable to models the relationship between one independent variable and the dependent variable of dichotomous nature like the fault-proneness in this study which always take either zero or one values. So, univariate logistic regression analysis is used to build the classification models for the fault-proneness of the classes of the six systems under study. This analysis is conducted to determine how well we can predict the fault-proneness of classes using only the coding standard violations-based metrics. To be able to evaluate the prediction power of the independent variables which are in this case study, the coding standard violations-based metrics, the univariate logistic regression analysis for each metric is applied separately (i.e. each time only one metric is taken as independent

variable with the fault-proneness as dependent variable. The univariate logistic regression is based on the following equation:

$$\hat{y} = \pi(x) \qquad (6.3)$$

$$y = \pi(x) + \varepsilon \qquad (6.4)$$

$$\pi(x) = \frac{e^{b_0 + b_1 x}}{1 + e^{b_0 + b_1 x}} \qquad (6.5)$$

Where $\hat{y}$ is the dependent variable to be predicted, y is the actual value of the dependent, π is the probability that a class will have faults, ε is the error in the prediction of the model, x is the independent variable, b_0 and b_1 are known as the model coefficients or parameters.

In logistic regression the dependent variable is not measured directly. The following simple example illustrates this concept. It might have a prediction model for the fault-proneness of classes as a function of the percentage of standard's rule violated in the class (PSRV). A result such as $\boldsymbol{\pi}$ (PSRV= 28) = 0.7 could be interpreted as "there is a 70% probability that a class with PSRV= 28 will be a faulty class." If the probability is 0.5 or more, the class will be classified as fault-prone. Otherwise, the class will be classified as non-fault-prone. Thus, in this case the class will be classified as fault-prone.

6.3.5.1.2 Univariate Linear Regression Models

Univariate linear regression (ULR) is a simple and useful technique for predicting a quantitative response. "Though it may seem somewhat dull compared to some of the more modern statistical learning approaches, univariate linear regression is still a useful and widely used statistical learning method" [41].

It is a very straightforward technique for predicting a quantitative response *Y* (dependent variable) on the basis of a single predictor variable (independent variable) *X*. it is an approach for modeling the relationship between a scalar dependent variable *Y* and one explanatory variable denoted *X* by fitting a linear equation to the observed data. This research used univariate linear regression to model the relationship between each coding standards violations-based metric (independent variable) and the faults density (dependent variable).

It assumes that there is approximately a linear relationship between the independent variable *X* and the dependent variable *Y*. Mathematically, this linear relationship can be written as:

$$\hat{y} = b_0 + b_1 x \qquad (6.6)$$

$$y = b_0 + b_1 x + \varepsilon \qquad (6.7)$$

Where x is the independent variable, b_0 and b_1 are two unknown constants that represent the intercept and slope terms in the linear model. Together, b_0 and b_1 are known as the model coefficients or parameters. $\hat{y}$ is the dependent variable to be predicted, y is the actual value of the dependent variable, and ε is the error in the prediction of the model.

6.3.5.2 Building Multivariate Prediction Models

As mentioned before, this research problem is of two types, classification and regression. So two types of multivariate regression models are built, multivariate logistic regression models and multivariate linear regression models. These models are built to predict the fault-proneness and the fault density of the classes, respectively.

6.3.5.2.1 Multivariate Logistic Regression Models

Multivariate logistic regression is useful for situations in which you want to be able to predict the presence or absence of a characteristic or outcome based on values of a set of predictor variables [38, 46, 47]. This means that those types of models are useful and suitable to models the relationship between many independent variables and the dependent variable of dichotomous nature like the fault-proneness in this study which always take either zero or one values. So, a multivariate logistic regression analysis is used to build the classification models for the fault-proneness of the classes of the six systems under study. This analysis is conducted to determine how well the fault-proneness of the classes can be predicted when using only the coding standard violations-based metrics, using only the CK metrics, and using a combination of both suites. As mentioned earlier, having too many independent variables may lead to high estimated standard error which, in turn, makes the built model data dependent and less generalizable. Since this research has a relatively big set of independent variables, an attribute selection strategy is applied to reduce the number of independent variables and remove the collinearity in each model. The multivariate logistic regression is based on this equation:

$$\hat{y} = \pi(x) \tag{6.8}$$

$$y = \pi(x) + \varepsilon \tag{6.9}$$

$$\pi(x_1, x_2, \dots, x_n) = \frac{e^{b_0 + b_1 x_1 + \cdots + b_n x_n}}{1 + e^{b_0 + b_1 x_1 + \cdots + b_n x_n}} \tag{6.10}$$

Where $\hat{y}$ is the predicted dependent variable, y is the actual dependent variable, π is the probability that a class will have faults. $x_1, x_2, \ldots, x_n$ are the independent variables. $b_0, b_1 \ldots b_n$ are the parameters to be estimated and ε is the error in the prediction.

It is worth here to mention that logistic regression does not measure the dependent variable directly in contrast to linear regression which measures it directly. The following simple example illustrates this concept. It might have a prediction model for the fault-proneness of classes as a function of just one variable, e.g., the percentage of standard's rules violated in the class (PSRV). A result such as π (PSRV= 28) = 0.7 could be interpreted as "there is a 70% probability that a class with PSRV= 28 will have faults." If the probability is 0.5 or more, the class will be classified as fault-prone. Otherwise, the class will be classified as non-fault-prone. Thus, in this case the class will be classified as fault-prone.

6.3.5.2.2 Multivariate Linear Regression Models

Multivariate linear regression (MLR) is the most commonly used technique for modeling the relationship between two or more independent variables and a dependent variable by fitting a linear equation to the observed data. The main advantages of this technique are its simplicity and that it is supported by many popular statistical packages. The general form of a MLR model can be given by:

$$\hat{y}_i = b_0 + b_1 x_{i1} + b_2 x_{i2} + \cdots + b_k x_{ik} \qquad (6.11)$$

$$y_i = b_0 + b_1 x_{i1} + b_2 x_{i2} + \cdots + b_k x_{ik} + \varepsilon_i \qquad (6.12)$$

Where $x_{i1}, \ldots, x_{ik}$ are the independent variables, b_i is the estimated regression coefficient, $\hat{y}_i$ is the dependent variable to be predicted, y_i is the actual value of the dependent variable, and ε_i is the error in the prediction of the *ith* case.

Multivariate linear regression is an approach for modeling the relationship between a scalar dependent variable y and two or more explanatory variables denoted *X* by fitting a linear equation to the observed data. This research used multivariate linear regression to build the prediction models for the fault density of the classes of the six systems under study. This analysis is conducted to determine how well the fault density of classes can be predicted when using only the coding standard violations-based metrics, using only the CK metrics, and using a combination of both suites. It is discussed in the previous section that having too many independent variables may lead to high estimated standard error which, in turn, makes the built model data dependent and less generalizable. Since this research has a relatively big set of independent variables, an attribute selection strategy is applied to reduce the number of independent variables and remove the collinearity in each model.

6.3.6 Measuring the Goodness of Fit

R-squared (R^2), also known as coefficient of determination, is probably the most popular measure of fit in statistical modeling. The mathematical formula for R^2 is given as follows.

$$R^2 = \frac{SS_{Reg}}{SS_{Total}} = 1 - \frac{SS_{Err}}{SS_{Total}} \qquad (6.13)$$

Where SS_{Reg}. is the sum of squares due to regression, SS_{err} is the residual sum of squares, and SS_{Total} is the total sum of squares.

However, this measure of goodness, is only appropriate for the linear regression model, but not for the logistic regression model. This yield a natural appeal for a measure that can be computed for a fitted model, is similar to R^2, takes values between 0 and 1,

becomes larger as the model "fits better", and provides a simple and clear interpretation [48]. Thus, researchers introduced what is called pseudo-R^2. There are several pseudo R^2s. In this research Nagelkerke R^2 and Cox & Snell R^2 are used. Nagelkerke R^2 is analogous to ordinary R^2 in the sense that it is on a similar scale, ranging from 0 to 1 (though some pseudo R^2 never achieve 0 or 1) with higher values indicating better model fit. The formula for Nagelkerke is given as follows.

$$Nagelkerke\ R^2 = \frac{1 - \left[\frac{L(M_0)}{L(M_B)}\right]^{\frac{2}{n}}}{1 - (L(M_0))^{\frac{2}{n}}} \qquad (6.14)$$

Where M_0 is the model without predictors, M_B is the model with predictors, and L is the estimated likelihood. Cox & Snell R^2 is referred to as a "pseudo-R" statistic, in that it is designed to tell us something similar to what R^2 tells us in ordinary least-squares regression, that of the proportion of variance accounted for in the dependent variable based on the predictive power of the independent variables (predictors) in the model. However, it should never be interpreted exactly as one would interpret R-squared in OLS (ordinary least-squares) regression. The formula for Cox & Snell R^2 is given as follows.

$$Cox\ \&\ Snell\ R^2 = 1 - \left(\frac{-2LL_{null}}{-2LL_k}\right)^{\frac{2}{n}} \qquad (6.15)$$

Where $-2LL_{null}$ is the loglikelihood for the empty model, and $-2LL_k$ is the loglikelihood for the model with the independent variables.

Other additional measures for the goodness of fit are the adjusted R^2 and -2Log likelihood (-2LL).

Adjusted R^2: is a modification of R^2 that adjusts for the number of explanatory terms in a model. Unlike R^2, the adjusted R^2 increases only if the new term improves the model

more than would be expected by chance. The adjusted R^2 can be negative, and will always be less than or equal to R^2. The mathematical formula for adjusted R2 is given as follows [41].

$$adjusted\ R^2 = 1 - \frac{RSS/(n-d-1)}{TSS/(n-1)} \qquad (6.16)$$

Where *TSS* is the total sum of squares and *RSS* is residual sum of squares. And *d* is the number of independent variables.

-2Log likelihood (-2LL): The -2LL for a model indicates the extent to which the model fails to perfectly predict the values of the dependent variable, i.e. it is a badness-of-fit indicator with large numbers mean poor fit of the model to the data. -2LL is analogous to the Error Sums of Squares, SSE, in OLS regression.

6.3.7 Validating the Regression Models

The explanatory model might fit the data very well (i.e. produce very small error). However, this might happen by pure chance. The cross-validation technique is applied to reduce this potential threat.

Cross-validation is a way of obtaining a realistic estimate of the predictive power of a model when it is applied to data sets other than those from which the model was derived. In general, a data set is divided into two subsets: training and testing sets. The training set is then used to fit or train the model while the testing set is used to test or validate the model. This is called split-sample validation [49]. Another alternative technique called V-fold cross-validation, which is a way of obtaining nearly unbiased estimators of prediction error. For a data set with ***n*** observations, a ***v***-fold cross-validation

divides the data set into ***v*** approximately equal partitions (folds), and each in turn is used for testing while the remaining are used for training. In our study, we set v=10 to be 10-fold cross validation in which the learning model is trained and tested 10 times. In this research, the learning model is trained on n/10 folds and tested on 1 fold. The cross-validation estimate of accuracy is the overall number of correct classifications divided by the number of instances (n) in the dataset [50].

The reasons behind the selection for this variant of v-fold cross-validation are: First, as pointed out by Myrtveit et al. [51], cross-validation is a widely used variant of v-fold cross-validation. Second, the greatest portion of dataset is used to train the model in each case, which in turn contributes in increasing the chance of getting as accurate estimate as possible [38, 45]. Third, the procedure is deterministic: no random sampling is involved[45].

6.3.8 Prediction Accuracy Measures

It is important to know, compared to other competitive models, how accurate any prediction model is. This can be done using some accuracy measures. Various measures of accuracy have been used in literature. For the logistic regression models, the correct classification rate and the receiver operating characteristic (ROC) area are used. ROC curve area or AUC expresses to which extent the built model is capable to correctly classify classes based on the considered functional correctness variable. "A 100% ROC area represents a perfect model that correctly classifies all classes, and larger ROC areas indicate that the model is better at classifying classes" [46]. This research work applied the following general rules to assess the classification performance according to the criteria adopted by [46] for AUC value: "AUC = 0.5 means that the classification is not

good, 0.5 < AUC < 0.6 means that the classification is poor, 0.6 ≤ AUC < 0.7 means that the classification is fair, 0.7 ≤ AUC < 0.8 means that the classification is acceptable, 0.8 ≤ AUC < 0.9 means that the classification is excellent, and AUC≥0.9 means that the classification is outstanding". "Thresholds based on the ROC analysis for the selected measures are considered practical if they fall at least within the acceptable range" [52].

It is worth here to mention that this research evaluate the prediction accuracy for logistic regression models based on the ROC curve area measure and the usage of correct classification rate is only to confirm the obtained results and to give more confidence about those obtained results.

Regarding the linear regression models, to specify to which extent the built models are accurate, this study used the mean absolute error (MAE) and the root mean squared error (RMSE) measures. These measures are based on what so called residual (that is, the difference between the predicted and the observed value). Suppose the testing set consist of n observations. Given an observation i, the mathematical formulas of the two measures are as follows.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n} \qquad (6.17)$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (y_{obs,i} - y_{model,i})^2}{n}} \qquad (6.18)$$

Where, $\mathbf{y_i}$ is the actual value of the *ith* case, $\hat{\mathbf{y}}_\mathbf{i}$ is the predicted value of the *ith* case and $\mathbf{y_{obs}}$ is observed values and $\mathbf{y_{model}}$ is modeled values at time/place ***i***.

CHAPTER 7

EMPIRICAL RESULTS AND ANALYSIS

This chapter presents the empirical results and analysis of the metrics investigated in this study as potential indicators for both the fault-proneness and the faults density of classes. In section 7.1, general analysis is presented. Univariate and multivariate analysis are presented in section 7.2 and 7.3 respectively. Section 7.4 presents the threats to validity.

7.1 General Analysis

This Section presents general analysis about the metrics investigated in this research.

7.1.1 Descriptive Statistics

This Section discusses the descriptive statistics for the metrics investigated in this study (both CK and the coding standard violations-based metrics) for the six systems under study (Synapse, Velocity, Poi, Xalan, Camel and Ant). As shown in Tables 7.1, 7.2, 7.3, 7.4, 7.5, 7.6. In each system, the 0s of the median of the corresponding metric indicate that more than 50% of system's classes do not violate any coding rule of the underlying category of rules measured by that metric. As shown in the Tables, 15, 16, 12, 12, 16, 14 metrics out of 24 metrics with 0s median for Synapse, Velocity, Poi, Xalan, Camel and Ant respectively.

Moreover, the third quartile (75%) of the PSRV and PSRVD metrics for all systems under study indicate that more than 75% of the system's classes have violations for at least one rule from the chosen standard for this study. Additionally, the third quartile (75%) of the PNCRV and PNCRVD metrics for all systems indicates that also more than 75% of the system's classes violated at least one of the naming conventions or rules of the underlying standard. Furthermore, the third quartile (75%) values for PSCV and PSCVD metrics for all systems indicate that more than 75% of each system's classes have violations for at least one category from the chosen standard for this study. On the other hand, the zero values of the third quartile 75% for PMCRV, PMCRVD, PTCRV, PTCRVD, PExcCRV, PExcCRVD, PConCRV and PConCRVD metrics indicate that less than 25% of each system's classes have violations for at least one rule of the corresponding category from the chosen standard for this study.

Generally, it can be observed that the naming, expressions and complexity coding rules' categories are the most frequent rules violated across all systems under study as explained by the median values. On the other hand, the least frequent coding rules violated across all systems are the coding rules of concurrency and exceptions categories.

From the descriptive statistics of CK metrics, it can be observed that all systems have almost the same level of coupling as explained by the mean and median values of the coupling metrics. Additionally, it can be observed that Xalan and Poi are more cohesive than the other systems as explained by the median and means of the cohesion metrics. Regarding the complexity, the classes of Poi, Xalan and Ant are more complex than the classes of the other three systems as explained by the descriptive statistics of

WMC. The mean and the maximum values of DIT in both case studies indicate that there is little reuse through inheritance.

Table 7.1: Metrics descriptive statistics for Synapse system

	PSRV	PSRVD	PNCRV	PNCRVD	PPCICRV	PPCICRVD	PFCRV	PFCRVD	PMCRV	PMCRVD	PDSCRV	PDSCRVD	PExpCRV	PExpCRVD	PExcCRV
MAX	25.58	1.16	100	12.50	25.00	1.56	50.00	1.79	20.00	0.35	40.00	1.18	33.33	0.79	50.00
MIN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	4.65	0.09	50.00	0.44	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MED	6.98	0.14	50.00	0.86	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
75%	11.63	0.25	50.00	1.72	0.00	0.00	25.00	0.27	0.00	0.00	0.00	0.00	11.11	0.11	0.00
MEAN	8.17	0.19	55.66	1.52	2.10	0.05	13.09	0.18	0.31	0.00	3.75	0.05	5.95	0.07	0.59
SD	4.49	0.17	28.99	1.93	5.18	0.21	15.17	0.29	2.49	0.02	8.59	0.16	7.68	0.12	5.39
	PExcCRVD	PTCRV	PTCRVD	PConCRV	PConCRVD	PComCRV	PComCRVD	PSCV	PSCVD	WMC	DIT	NOC	CBO	RFC	LCOM
MAX	0.45	25.00	1.79	0.00	0.00	100	50.00	80.00	5.00	67.00	5.00	19	83.00	172	1931
MIN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0	1	0	0	0	0
25%	0.00	0.00	0.00	0.00	0.00	100	0.31	20.00	0.33	3.00	1.00	0.00	7.00	12.00	1.00
MED	0.00	0.00	0.00	0.00	0.00	100	1.04	30.00	0.58	4.00	1.00	0.00	10.00	23.00	3.00
75%	0.00	0.00	0.00	0.00	0.00	100	2.22	40.00	1.00	8.00	2.00	0.00	14.00	38.25	11.25
MEAN	0.00	0.78	0.01	0.00	0.00	76.56	1.96	30.82	0.79	7.62	1.64	0.41	12.75	29.53	41.09
SD	0.03	4.36	0.12	0.00	0.00	42.44	3.75	14.35	0.73	9.11	0.78	2.19	11.74	25.83	175.82

Table 7.2: Metrics descriptive statistics for Velocity system

	PSRV	PSRVD	PNCRV	PNCRVD	PPCICRV	PPCICRVD	PFCRV	PFCRVD	PMCRV	PMCRVD	PDSCRV	PDSCRVD	PExpCRV	PExpCRVD	PExcCRV
MAX	37.21	1.55	100	16.67	25	0.227	50	4.167	20	0.645	80	3.33	44.44	0.585	50
MIN	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25%	4.65	0.08	50.00	0.31	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MED	6.98	0.14	50.00	0.81	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
75%	11.63	0.22	50.00	2.00	0.00	0.00	25.00	0.20	0.00	0.00	20.00	0.06	11.11	0.10	0.00
MEAN	8.20	0.17	53.71	1.37	1.04	0.01	11.79	0.17	1.48	0.01	6.90	0.11	6.16	0.06	0.66
SD	5.79	0.15	31.37	1.80	3.83	0.03	16.31	0.39	5.25	0.06	12.82	0.32	9.32	0.11	5.70
	PExcCRVD	PTCRV	PTCRVD	PConCRV	PConCRVD	PComCRV	PComCRVD	PSCV	PSCVD	WMC	DIT	NOC	CBO	RFC	LCOM
MAX	0.962	25	0.676	0	0	100	16.667	90	5	153	5	39	80	250	8092
MIN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
25%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	20.00	0.28	3.00	1.00	0.00	4.00	6.00	0.00
MED	0.00	0.00	0.00	0.00	0.00	100	0.46	30.00	0.49	5.00	1.00	0.00	7.00	14.00	3.00
75%	0.00	0.00	0.00	0.00	0.00	100	1.52	40.00	0.93	9.00	2.00	0.00	11.00	30.00	10.00
MEAN	0.01	1.53	0.02	0.00	0.00	58.95	1.00	29.39	0.67	9.02	1.68	0.44	10.81	22.98	80.34
SD	0.07	6.00	0.07	0.00	0.00	49.30	1.60	17.05	0.57	14.14	0.89	2.75	12.72	27.37	554.87

Table 7.3: Metrics descriptive statistics for Poi system

	PSRV	PSRVD	PNCRV	PNCRVD	PPCICRV	PPCICRVD	PFCRV	PFCRVD	PMCRV	PMCRVD	PDSCRV	PDSCRVD	PExpCRV	PExpCRVD	PExcCRV
MAX	34.88	0.93	100	10.00	50.00	1.14	75.00	1.19	20.00	0.38	80.00	2.00	44.44	0.65	50.00
MIN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	6.98	0.08	50.00	0.30	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MED	11.63	0.14	50.00	0.73	12.50	0.04	0.00	0.00	0.00	0.00	20.00	0.12	11.11	0.08	0.00
75%	16.28	0.20	100	1.09	12.50	0.19	25.00	0.15	0.00	0.00	20.00	0.30	11.11	0.16	0.00
MEAN	11.20	0.15	59.79	0.95	8.91	0.12	11.56	0.11	0.96	0.01	14.90	0.17	8.81	0.10	0.23
SD	5.96	0.10	31.62	1.18	9.25	0.16	15.99	0.21	4.27	0.03	13.43	0.21	7.47	0.10	3.37
	PExcCRVD	PTCRV	PTCRVD	PConCRV	PConCRVD	PComCRV	PComCRVD	PSCV	PSCVD	WMC	DIT	NOC	CBO	RFC	LCOM
MAX	0.56	50.00	3.13	0.00	0.00	100	4.76	80.00	4.00	134	6.00	134	214	390	7059
MIN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
25%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	30.00	0.31	5.00	1.00	0.00	5.00	13.00	2.00
MED	0.00	0.00	0.00	0.00	0.00	0.00	0.00	40.00	0.52	10.00	2.00	0.00	6.00	21.00	24.00
75%	0.00	0.00	0.00	0.00	0.00	100	0.87	50.00	0.76	16.00	2.00	0.00	9.00	36.50	53.50
MEAN	0.00	1.08	0.03	0.00	0.00	47.61	0.56	40.18	0.58	13.58	1.87	0.74	10.18	30.51	101.15
SD	0.03	5.37	0.28	0.00	0.00	50.00	0.88	17.80	0.41	14.70	0.85	6.99	19.64	37.14	443.28

Table 7.4: Metrics descriptive statistics for Xalan system

	PSRV	PSRVD	PNCRV	PNCRVD	PPCICRV	PPCICRVD	PFCRV	PFCRVD	PMCRV	PMCRVD	PDSCRV	PDSCRVD	PExpCRV	PExpCRVD	PExcCRV
MAX	34.88	4.65	100	25.00	50.00	1.19	75.00	2.50	60.00	2.00	100	4.00	44.44	11.11	50.00
MIN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	4.65	0.05	50.00	0.12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MED	9.30	0.11	50.00	0.41	0.00	0.00	0.00	0.00	0.00	0.00	20.00	0.05	11.11	0.01	0.00
75%	13.95	0.18	50.00	0.98	0.00	0.00	25.00	0.17	0.00	0.00	20.00	0.20	11.11	0.11	0.00
MEAN	10.50	0.15	44.00	0.93	3.20	0.02	13.77	0.12	3.06	0.02	16.05	0.16	8.88	0.10	1.71
SD	7.09	0.21	19.75	1.73	6.59	0.09	18.03	0.24	7.76	0.09	17.31	0.32	10.67	0.43	9.10
	PExcCRVD	PTCRV	PTCRVD	PConCRV	PConCRVD	PComCRV	PComCRVD	PSCV	PSCVD	WMC	DIT	NOC	CBO	RFC	LCOM
MAX	0.66	50.00	2.08	33.33	0.37	100	20.00	90.00	20.00	133	8.00	29	168	409	7774
MIN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
25%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	20.00	0.19	3.00	1.00	0.00	2.00	7.00	0.00
MED	0.00	0.00	0.00	0.00	0.00	100	0.37	40.00	0.40	5.00	3.00	0.00	6.00	17.00	3.00
75%	0.00	0.00	0.00	0.00	0.00	100	1.18	50.00	0.70	11.00	3.00	0.00	14.50	39.00	28.00
MEAN	0.01	5.49	0.07	0.08	0.00	68.23	0.92	38.03	0.58	11.12	2.52	0.52	12.17	29.54	125.25
SD	0.05	10.76	0.23	1.59	0.01	46.59	1.58	20.80	0.88	16.28	1.44	2.33	17.92	37.22	584.21

Table 7.5: Metrics descriptive statistics for Camel system

	PSRV	PSRVD	PNCRV	PNCRVD	PPCICRV	PPCICRVD	PFCRV	PFCRVD	PMCRV	PMCRVD	PDSCRV	PDSCRVD	PExpCRV	PExpCRVD	PExcCRV
MAX	23.26	0.93	100	12.50	37.50	1.14	75.00	2.27	40.00	0.83	20.00	2.22	44.44	0.86	50.00
MIN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	2.33	0.08	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MED	4.65	0.14	50.00	0.64	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
75%	9.30	0.22	100	1.47	0.00	0.00	25.00	0.26	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MEAN	6.49	0.16	45.12	1.06	0.94	0.01	11.04	0.18	0.73	0.01	4.16	0.09	3.20	0.04	0.05
SD	4.62	0.12	38.76	1.47	3.63	0.07	16.70	0.34	3.86	0.05	8.12	0.25	6.40	0.10	1.64
	PExcCRVD	PTCRV	PTCRVD	PConCRV	PConCRVD	PComCRV	PComCRVD	PSCV	PSCVD	WMC	DIT	NOC	CBO	RFC	LCOM
MAX	0.53	25.00	1.39	0.00	0.00	100	16.67	70.00	4.00	166	6.00	39	448	322	13617
MIN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	10.00	0.30	3.00	1.00	0.00	4.00	6.00	0.00
MED	0.00	0.00	0.00	0.00	0.00	100	0.36	20.00	0.53	5.00	1.00	0.00	7.00	15.00	4.00
75%	0.00	0.00	0.00	0.00	0.00	100	1.54	30.00	0.87	11	3.00	0.00	12.00	28.00	28.00
MEAN	0.00	0.24	0.00	0.00	0.00	52.52	1.07	23.74	0.64	8.59	1.97	0.53	11.14	21.74	80.07
SD	0.02	2.44	0.06	0.00	0.00	49.96	1.72	14.64	0.51	11.30	1.29	2.64	22.85	25.17	531.99

Table 7.6: Metrics descriptive statistics for Ant system

	PSRV	PSRVD	PNCRV	PNCRVD	PPCICRV	PPCICRVD	PFCRV	PFCRVD	PMCRV	PMCRVD	PDSCRV	PDSCRVD	PExpCRV	PExpCRVD	PExcCRV
MAX	30.23	0.93	100	16.67	25.00	0.74	75.00	2.27	40.00	0.77	60.00	4.00	44.44	1.59	50.00
MIN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	4.65	0.07	50.00	0.24	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MED	9.30	0.11	50.00	0.69	0.00	0.00	25.00	0.09	0.00	0.00	0.00	0.00	0.00	0.00	0.00
75%	13.95	0.18	100	1.32	0.00	0.00	25.00	0.28	0.00	0.00	20.00	0.05	11.11	0.11	0.00
MEAN	9.96	0.14	60.73	1.10	1.89	0.02	18.39	0.19	1.97	0.01	6.02	0.09	8.01	0.08	1.15
SD	6.85	0.11	36.34	1.57	4.93	0.06	19.06	0.29	6.05	0.06	9.97	0.28	9.86	0.15	7.49
	PExcCRVD	PTCRV	PTCRVD	PConCRV	PConCRVD	PComCRV	PComCRVD	PSCV	PSCVD	WMC	DIT	NOC	CBO	RFC	LCOM
MAX	1.92	50.00	2.50	33.33	0.16	100	14.29	90.00	4.00	120	7.00	102	499	288	6692
MIN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
25%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	20.00	0.22	4.00	1.00	0.00	4.00	11.00	0.00
MED	0.00	0.00	0.00	0.00	0.00	100	0.64	30.00	0.41	7.00	2.00	0.00	7.00	24.00	6.00
75%	0.00	25.00	0.03	0.00	0.00	100	1.47	50.00	0.70	14.00	4.00	0.00	11.00	43.00	53.00
MEAN	0.01	6.51	0.06	0.04	0.00	71.12	1.05	34.20	0.53	11.11	2.53	0.74	11.10	34.50	89.62
SD	0.10	11.21	0.18	1.22	0.01	45.35	1.45	20.13	0.48	11.99	1.40	4.81	26.40	36.07	350.82

Table 7.7 shows descriptive statistics for classes' Size in terms of LOC in all systems under study.

Table 7.7: Size descriptive statistics for classes' LOC of all systems under study.

	synapse	Velocit	Poi	Xalan	Camel	Ant
MAX	536.00	4444.00	1308.00	2639.00	932.00	1399.00
MIN	2.00	4.00	4.00	1.00	2.00	3.00
25%	19.00	19.00	43.50	30.00	14.00	28.00
MED	49.00	44.00	69.00	75.00	35.00	66.00
75%	102.25	110.00	135.00	199.00	77.00	136.00
MEAN	75.99	110.22	117.49	173.13	60.55	118.19
SD	79.36	359.05	154.17	257.39	78.79	157.10

7.1.2 Principle Component Analysis

This Section presents the results from principal component analysis for the six systems considered by this study using only the coding standard violations-based metrics data. As shown in Tables C.1, C.2, C.3, C.4, C.5, C.6 and C.7 in the Appendix C, the coding standard violations-based metrics measures between 9 to 11 dimensions.

7.1.2.1 Dimensions Captured by Coding Standard Violations-Based Metrics

The PCA results presented in Tables C.1, C.2, C.3, C.4, C.5, C.6 and C.7 in the Appendix C show that the dimensions captured by the coding standard violations-based metrics can be classified into the following dimensions. Standard's rules and categories, naming rules, classes and interfaces rules, fields rules, methods rules, types rules, declarations and statements rules, expressions rules, exceptions rules, concurrency rules and complexity rules. These dimensions reflect the standard rules' categories which the metrics are derived from.

The results in Tables 7.8 and 7.9 show some overlapping among these dimensions. For example, for some metrics, the expectation to fall into a certain dimension however, they fall into other dimensions. The general observation from Tables 7.8 and 7.9 is that metrics which were found to be significant are falling in the first two components in almost all case studies which in turn reflect the importance of these metrics. For instance the metrics PSRV and PSCV in all case studies fall into the first or the second component. Additionally it is clear from Tables 7.8 and 7.9 that except for the first two components, each component correspond to one dimension. For example, in camel case study, the PC3, PC4, PC5, PC6, PC7 and PC8 correspond to expression rules dimension, exceptions rules dimension, fields rules dimension, Methods rules dimension, declarations and statements rules dimension, packages and classes rules dimension, types rules dimension and complexity rules dimension respectively.

Table 7.8: PCA for coding standard violations-based metrics for (Ant, Velocity and Synapse)

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11
Ant	PSRV	PSRVD	PConCRV	PPCICRV	PMCRV	PExcCRV	PTCRV	PDSCRV	PComCRV	PExpCRVD	PFCRVD
	PNCRV	PNCRVD	PConCRVD	PPCICRVD	PMCRVD	PExcCRVD	PTCRVD	PDSCRVD	PComCRVD		
	PFCRV	PSCVD									
	PExpCRV										
	PSCV										
Velocity	PSRV	PSRVD	PTCRV	PMCRV	PExcCRV	PPCICRV	PExpCRV	PDSCRV			
	PNCRV	PNCRVD	PTCRVD	PMCRVD	PExcCRVD	PPCICRVD	PExpCRVD	PDSCRVD			
	PFCRV	PFCRVD									
	PComCRV	PComCRVD									
	PSCV	PSCVD									
Synapse	PSRVD	PSRV	PFCRVD	PExcCRV	PMCRV	PDSCRV	PExpCRV	PPCICRV	PComCRV		
	PNCRVD	PNCRV	PTCRV	PExcCRVD	PMCRVD	PDSCRVD	PExpCRVD	PPCICRVD	PComCRVD		
	PSCVD	PFCRV	PTCRVD								
		PSCV									

Table 7.9: PCA for coding standard violations-based metrics for (Poi, Xalan , Camel and all systems)

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11
Poi	PSRVD	PSRV	PDSCRV	PPCICRV	PMCRV	PExcCRV	PExpCRV	PComCRV	PTCRV		
	PNCRVD	PNCRV	PDSCRVD	PPCICRVD	PMCRVD	PExcCRVD	PExpCRVD	PComCRVD	PTCRVD		
	PSCVD	PFCRV									
		PFCRVD									
		PSCV									
Xalan	PSRV	PSRVD	PExcCRV	PConCRV	PPCICRV	PTCRV	PDSCRVD	PFCRV	PMCRVD	PNCRV	
	PMCRV	PExpCRVD	PExcCRVD	PConCRVD	PPCICRVD	PTCRVD	PComCRVD	PFCRVD		PNCRVD	
	PDSCRV	PSCVD									
	PExpCRV										
	PComCRV										
	PSCV										
Camel	PSRVD	PSRV	PExpCRV	PExcCRV	PFCRV	PMCRV	PDSCRV	PPCICRV	PTCRV	PComCRV	
	PNCRVD	PNCRV	PExpCRVD	PExcCRVD	PFCRVD	PMCRVD	PDSCRVD	PPCICRVD	PTCRVD	PComCRVD	
	PSCVD	PSCV									
All Systems	PSRV	PSRVD	PConCRV	PExcCRV	PPCICRV	PMCRV	PTCRV	PFCRV	PComCRV	PDSCRV	PNCRVD
	PNCRV	PExpCRVD	PConCRVD	PExcCRVD	PPCICRVD	PMCRVD	PTCRVD	PFCRVD	PComCRVD	PDSCRVD	
	PExpCRV	PSCVD									
	PSCV										

7.1.3 Bivariate Correlation Analysis

To explore the relationship between the proposed metrics and the functional correctness quality attribute in terms of fault-proneness and fault density, bivariate correlation analysis is performed as presented in Sections 7.1.3.1 and 7.1.3.2.

7.1.3.1 Bivariate Correlation Analysis with Respect To Fault-proneness

To explore the relationship between each metric of the coding standard violations-based suite and the fault-proneness, two correlation techniques were performed. First, the Spearman correlation coefficient is calculated between each metric and the binary variable capturing whether the class is fault prone or not. For each system from the target set of systems under study, the correlation has been obtained from the data of all system's classes. More precisely, from 741, 933, 875, 439, 229 and 256 data points of Ant, Camel, Xalan, Poi, Velocity and Synapse systems respectively. The results of correlation coefficients and p-values using spearman's technique are presented in Tables 7.10 and 7.11. In addition to Spearman's correlation analysis, we also performed Kendall's tau correlation analysis. "Kendall's tau correlation analysis adjusts for multiple observations (data points) having the same value and does not depend on the distribution of the data" [53]. The results in terms of correlation coefficient and p-values using Kendall's tau technique are presented in Tables A.1 and A.2 in the appendix A.

For each metric, the Hypothesis 1 was tested at 0.05 level of significance. The values that are rendered in boldface highlight significant correlation coefficients at 0.05 level as shown in Tables 7.10 and 7.11. It is clear to observe that PSRV, PNCRV, PFCRV, PExpCRV, PExpCRVD and PSCV were found to be significantly correlated with the fault proneness of classes across all the systems under study. Regarding the rest

of metrics, the correlation analysis results show that PSRVD, PComCRV and PSCVD were found to be significantly correlated with fault proneness in all systems except Camel, Poi and Camel respectively. However, the correlation analysis results show that PPCICRVD, PDSCRV, PDSCRVD, PMCRV, PMCRVD, PNCRVD, PFCRVD, PPCICRV, PExcCRV, PExcCRVD, PTCRV, PTCRVD and PComCRVD were found to be significantly correlated with fault proneness in three or four systems from the target set of systems under study. This means that naming, expressions, complexity and fields coding rules' categories were found to be significantly correlated with fault proneness. So the hypothesis 1 is partially accepted and concludes that only some of the coding standard violations-based metrics are significantly correlated with the fault proneness of classes. The differences in the significance of correlation across the systems under study can be discussed as follows. The class code size in terms of lines of code (LOC without comments and blank lines) is a dominant factor that governs the number of introduced violations for coding standard's rules in addition to the diversity of such introduced violations. So the difference in size across system's classes has a great impact on the values of coding standard violations-based metrics which in turn, affect the correlation significance between the metrics under study and the fault proneness of classes. Table 7.7 presents more descriptive statistics about the classes' size for all systems investigated by this study.

Some common results can be observed from the six case studies. For example, the positive correlation between PSRV, PNCRV, PFCRV, PExpCRV, PExpCRVD and PSCV metrics and the class fault proneness suggest that the higher values for these metrics, the more the probability of being a faulty class. Additionally, it is observed that

PConCRV and PConCRVD reported to be blank p-values and correlation coefficients in Synapse, Velocity, Poi and Camel systems because of the zero values of all observations for these two metrics. This implies that either the classes of these systems do not violate any rules of the concurrency category or the systems nature is irrelative to parallelism and concurrency. Regarding Ant and Xalan systems, the correlation analysis show that PconCRV and PConCRVD were found to be insignificantly correlated with fault proneness. By inspecting the observations of these two systems, it was found that only two observations in Xalan and one observation in Ant violate the concurrency category which is considered neglectable with contrast to 875 and 741 observations of Xalan and Ant respectively.

Table 7.10 Spearman correlation coefficient with fault-proneness

	Synapse		Velocity		Poi	
Metric	Correlation coefficient	p-Value	Correlation coefficient	p-Value	Correlation coefficient	p-Value
PSRV	**0.418482**	0.000000	**0.275297**	0.000024	**0.492773**	0.000000
PSRVD	**-0.296095**	0.000001	**-0.239308**	0.000257	**-0.098001**	0.040126
PNCRV	**0.251412**	0.000047	**0.233806**	0.000359	**0.312124**	0.000000
PNCRVD	**-0.223738**	0.000308	-0.049452	0.456447	-0.088286	0.064584
PPCICRV	**0.207141**	0.000855	0.077794	0.240968	**0.207989**	0.000011
PPCICRVD	**0.184447**	0.003055	0.072374	0.275422	0.030419	0.524992
PFCRV	**0.358342**	0.000000	**0.182726**	0.005547	**0.207966**	0.000011
PFCRVD	**0.260391**	0.000025	0.112396	0.089709	**0.159921**	0.000771
PMCRV	0.110449	0.077743	0.112813	0.088518	0.078168	0.101915
PMCRVD	0.110705	0.077052	0.109635	0.097926	0.076717	0.108452
PDSCRV	**0.154251**	0.013483	0.014542	0.826758	**0.431590**	0.000000
PDSCRVD	**0.142077**	0.022987	-0.017459	0.792726	**0.281922**	0.000000
PExpCRV	**0.285134**	0.000004	**0.191983**	0.003540	**0.461917**	0.000000
PExpCRVD	**0.211864**	0.000645	**0.143271**	0.030203	**0.298806**	0.000000
PExcCRV	0.076250	0.224062	**0.160305**	0.015169	0.050478	0.291296
PExcCRVD	0.076849	0.220434	**0.160301**	0.015172	0.050478	0.291297
PTCRV	**0.157453**	0.011647	0.085823	0.195661	-0.013102	0.784279
PTCRVD	**0.157427**	0.011661	0.085263	0.198604	-0.015773	0.741730
PConCRV						
PConCRVD						
PComCRV	**0.198288**	0.001429	**0.243835**	0.000194	0.045152	0.345259
PComCRVD	-0.095578	0.127193	**0.133135**	0.044153	-0.061807	0.196166
PSCV	**0.396417**	0.000000	**0.278336**	0.000019	**0.504520**	0.000000
PSCVD	**-0.322739**	0.000000	**-0.229962**	0.000451	**-0.136950**	0.004043

Table 7.11 Spearman correlation coefficient with fault-proneness

	Xalan		Camel		Ant		All Systems	
Metric	Correlation coefficient	p-Value	Correlation coefficient	p-Value	Correlation coefficient	p-Value	Correlation coefficient	p-Value
PSRV	**0.288084**	0.000000	**0.193148**	0.000000	**0.433545**	0.000000	**0.371892**	0.000000
PSRVD	**-0.292258**	0.000000	0.012468	0.703701	**-0.248063**	0.000000	**-0.173660**	0.000000
PNCRV	**0.220669**	0.000000	**0.133363**	0.000044	**0.332704**	0.000000	**0.210129**	0.000000
PNCRVD	**-0.142265**	0.000024	**0.078420**	0.016583	**-0.170803**	0.000003	**-0.068726**	0.000050
PPCICRV	**0.250708**	0.000000	0.046141	0.159056	**0.237716**	0.000000	**0.271898**	0.000000
PPCICRVD	**0.220667**	0.000000	0.044782	0.171715	**0.217551**	0.000000	**0.248214**	0.000000
PFCRV	**0.110630**	0.001046	**0.148787**	0.000005	**0.340896**	0.000000	**0.177867**	0.000000
PFCRVD	0.020903	0.536903	**0.127833**	0.000090	**0.095927**	0.008978	**0.084917**	0.000001
PMCRV	**0.133953**	0.000070	**0.078392**	0.016622	**0.294054**	0.000000	**0.154335**	0.000000
PMCRVD	**0.119959**	0.000376	**0.077926**	0.017281	**0.280741**	0.000000	**0.148269**	0.000000
PDSCRV	**0.241808**	0.000000	0.040355	0.218141	**0.207646**	0.000000	**0.288882**	0.000000
PDSCRVD	0.064495	0.056516	0.030397	0.353691	**0.131341**	0.000337	**0.208056**	0.000000
PExpCRV	**0.214665**	0.000000	**0.164095**	0.000000	**0.373424**	0.000000	**0.318379**	0.000000
PExpCRVD	**0.087019**	0.010016	**0.148788**	0.000005	**0.181346**	0.000001	**0.230389**	0.000000
PExcCRV	-0.051489	0.128035	**0.065425**	0.045732	**0.155476**	0.000021	**0.045616**	0.007173
PExcCRVD	-0.051092	0.131007	**0.065425**	0.045732	**0.154295**	0.000025	**0.045472**	0.007359
PTCRV	**0.100523**	0.002913	0.005373	0.869814	**0.254679**	0.000000	**0.111212**	0.000000
PTCRVD	**0.068483**	0.042842	0.005373	0.869816	**0.203164**	0.000000	**0.099099**	0.000000
PConCRV	0.050856	0.132794			-0.019752	0.591398	0.019662	0.246690
PConCRVD	0.050856	0.132794			-0.019752	0.591398	0.019662	0.246690
PComCRV	**0.140569**	0.000030	**0.127551**	0.000094	**0.270971**	0.000000	**0.137609**	0.000000
PComCRVD	**-0.107414**	0.001463	**0.076989**	0.018674	-0.043532	0.236590	**-0.043965**	0.009562
PSCV	**0.299743**	0.000000	**0.197754**	0.000000	**0.416743**	0.000000	**0.384141**	0.000000
PSCVD	**-0.289325**	0.000000	-0.002636	0.935918	**-0.276185**	0.000000	**-0.186757**	0.000000

7.1.3.2 Bivariate Correlation Analysis With Respect To Fault Density

To explore the relationship between each metric of the coding standard violations-based suite and the fault density, two correlation techniques are performed. First, the Spearman correlation coefficient is calculated between each metric and the variable capturing the density of faults which defined as the number of faults in the class divided by the code size in terms of KLOC (excluding comments and blank lines). For each system from the target set of systems under study, the correlation has been obtained from the data of all system's classes. More precisely, from 741, 933, 875, 439, 229 and 256 data points of Ant, Camel, Xalan, Poi, Velocity and Synapse systems respectively. The results of correlation coefficients and p-values using spearman's technique are presented in Tables 7.12 and 7.13. In addition to Spearman's correlation analysis, we also performed Kendall's tau correlation analysis. "Kendall's tau correlation analysis adjusts for multiple observations (data points) having the same value and does not depend on the distribution of the data" [53]. The results in terms of correlation coefficient and p-values using Kendall's tau technique are presented in Tables A.3 and A.4 in the appendix A.

For each metric, the Hypothesis 5 was tested at 0.05 level of significance. The values that are rendered in boldface highlight significant correlation coefficients at 0.05 level as shown in Tables 7.12 and 7.13. It is clear to observe that PSRV, PNCRV, PExpCRV and PSCV were found to be significantly correlated with the fault density of classes across all the systems under study. Regarding the rest of metrics, the correlation analysis results show that PSCVD was found to be significantly correlated with fault density in all systems except Camel system. However, the correlation analysis results also show that PFCRV, PComCRV, PPCICRVD, PDSCRV, PDSCRVD, PNCRVD,

PFCRVD, PPCICRV, PExcCRV, PExcCRVD, PTCRV, PTCRVD and PComCRVD were found to be significantly correlated with fault density in two, three or four systems from the target set of systems under study. Furthermore, the correlation analysis results show that PMCRV and PMCRVD were found to be significantly correlated with fault density only in Ant system. So the hypothesis 5 is partially accepted and concludes that only some of the coding standard violations-based metrics are significantly correlated with the fault density of classes. The differences in the significance of correlation across the systems under study can be discussed as follows. The class code size in terms of lines of code (LOC without comments and blank lines) is a dominant factor which has a great impact on the number of introduced violations for coding standard's rules in addition to the diversity of such introduced violations. So the differences in size across system's classes might impact on the values of coding standard violations-based metrics which in turn, affect the correlation significance between the metrics under study and the fault density of classes. Table 7.7 presents more descriptive statistics about the classes' size for all systems investigated by this study.

Some common results can be observed from the six case studies. For example, the positive correlation between PSRV, PNCRV, PExpCRV, and PSCV metrics and the class fault density suggest that the higher values for these metrics, the more the faults density of the class. Additionally, it is observed that PConCRV and PConCRVD reported to be blank p-values and correlation coefficients in Synapse, Velocity, Poi and Camel systems because of the zero values of all observations for these two metrics. This implies that either the classes of these systems do not violate any rules of the concurrency category or the systems nature is irrelative to parallelism and concurrency. Regarding Ant and Xalan

systems, the correlation analysis show that PconCRV and PConCRVD were found to be insignificant correlated with fault density. By inspecting the observations of these two systems, only two observations in Xalan and one observation in Ant were found to violate the concurrency category which can be considered neglectable with contrast to 875 and 741 observations of Xalan and Ant respectively.

Table 7.12: Spearman correlation coefficient with respect to fault density for Synapse, Velocity and Poi systems

	Synapse		Velocity		Poi	
Metric	Correlation coefficient	p-Value	Correlation coefficient	p-Value	Correlation coefficient	p-Value
PSRV	**0.335492**	0.000000	**0.202353**	0.002089	**0.237077**	0.000001
PSRVD	**-0.197367**	0.001505	**-0.156261**	0.017968	**0.140554**	0.003165
PNCRV	**0.177659**	0.004353	**0.190348**	0.003838	**0.151404**	0.001465
PNCRVD	**-0.155052**	0.013001	0.019490	0.769255	0.084243	0.077867
PPCICRV	**0.156289**	0.012287	0.034126	0.607429	0.065997	0.167480
PPCICRVD	**0.141870**	0.023189	0.031219	0.638388	0.023757	0.619603
PFCRV	**0.298831**	0.000001	0.106232	0.108865	0.041167	0.389541
PFCRVD	**0.248558**	0.000058	0.064914	0.328084	0.061646	0.197342
PMCRV	0.086922	0.165574	0.051018	0.442294	-0.012545	0.793238
PMCRVD	0.087438	0.163066	0.049948	0.451943	-0.012932	0.787013
PDSCRV	0.104723	0.094532	-0.038563	0.561520	**0.277302**	0.000000
PDSCRVD	0.101283	0.105930	-0.054306	0.413409	**0.297519**	0.000000
PExpCRV	**0.213806**	0.000573	**0.132107**	0.045830	**0.256545**	0.000000
PExpCRVD	**0.177616**	0.004363	0.106232	0.108864	**0.292902**	0.000000
PExcCRV	0.039133	0.533081	0.124849	0.059247	0.028554	0.550716
PExcCRVD	0.039628	0.527917	0.124969	0.059001	0.028685	0.548889
PTCRV	**0.178464**	0.004177	0.087167	0.188726	0.025995	0.587000
PTCRVD	**0.179435**	0.003973	0.088306	0.182990	0.023862	0.618051
PConCRV						
PConCRVD						
PComCRV	**0.143414**	0.021717	**0.179509**	0.006454	-0.070243	0.141729
PComCRVD	-0.060543	0.334625	0.123753	0.061529	-0.081814	0.086866
PSCV	**0.322136**	0.000000	**0.208066**	0.001545	**0.257937**	0.000000
PSCVD	**-0.218508**	0.000429	**-0.140332**	0.033795	**0.127652**	0.007407

Table 7.13: Spearman correlation coefficient with respect to fault density for Xalan, Camel, Ant and all system

	Xalan		Camel		Ant		All Systems	
Metric	Correlation coefficient	p-Value	Correlation coefficient	p-Value	Correlation coefficient	p-Value	Correlation coefficient	p-Value
PSRV	**0.127549**	0.000155	**0.161991**	0.000001	**0.397568**	0.000000	**0.281789**	0.000000
PSRVD	**-0.105490**	0.001780	0.038488	0.240208	**-0.212604**	0.000000	**-0.083813**	0.000001
PNCRV	**0.155334**	0.000004	**0.109456**	0.000811	**0.312378**	0.000000	**0.171756**	0.000000
PNCRVD	0.008763	0.795760	**0.080648**	0.013735	**-0.140713**	0.000122	-0.005250	0.757096
PPCICRV	**0.102247**	0.002461	0.035374	0.280417	**0.214074**	0.000000	**0.203999**	0.000000
PPCICRVD	**0.094547**	0.005126	0.034588	0.291246	**0.196841**	0.000000	**0.190923**	0.000000
PFCRV	0.029775	0.379029	**0.126776**	0.000103	**0.307481**	0.000000	**0.121038**	0.000000
PFCRVD	-0.006003	0.859265	**0.115186**	0.000423	**0.093028**	0.011291	**0.065508**	0.000112
PMCRV	0.039536	0.242690	0.055124	0.092418	**0.264723**	0.000000	**0.093032**	0.000000
PMCRVD	0.036518	0.280580	0.054813	0.094275	**0.253324**	0.000000	**0.089837**	0.000000
PDSCRV	**0.105975**	0.001694	0.030386	0.353873	**0.189935**	0.000000	**0.216592**	0.000000
PDSCRVD	0.043298	0.200703	0.022634	0.489871	**0.121479**	0.000921	**0.171345**	0.000000
PExpCRV	**0.072167**	0.032806	**0.131926**	0.000053	**0.339774**	0.000000	**0.236120**	0.000000
PExpCRVD	0.048719	0.149885	**0.120291**	0.000231	**0.174149**	0.000002	**0.194909**	0.000000
PExcCRV	-0.064710	0.055696	**0.076016**	0.020224	**0.143522**	0.000088	0.029045	0.086999
PExcCRVD	-0.063704	0.059618	**0.076016**	0.020224	**0.142528**	0.000099	0.029080	0.086622
PTCRV	0.047456	0.160751	0.001485	0.963870	**0.228464**	0.000000	**0.067092**	0.000076
PTCRVD	0.032719	0.333688	0.001545	0.962414	**0.183596**	0.000000	**0.059847**	0.000417
PConCRV	0.033999	0.315115			-0.019542	0.595338	0.011848	0.485175
PConCRVD	0.034021	0.314799			-0.019542	0.595338	0.011851	0.485062
PComCRV	0.022767	0.501218	**0.102532**	0.001713	**0.256968**	0.000000	**0.080128**	0.000002
PComCRVD	**-0.085180**	0.011714	**0.068395**	0.036729	-0.023317	0.526253	**-0.035682**	0.035487
PSCV	**0.133914**	0.000071	**0.167670**	0.000000	**0.382163**	0.000000	**0.292529**	0.000000
PSCVD	**-0.094460**	0.005167	0.026092	0.426001	**-0.239620**	0.000000	**-0.092186**	0.000000

7.2 Univariate Regression Analysis Results

For each one of the target set of systems under study, two families of univariate regression models are built. One of them for the fault proneness dependent variable while the other for the fault density dependent variable. Each one of those families comprises a single univariate regression model for each variable from the independent variables set which in this research are the coding standard violations-based metrics. The purpose of this analysis is to investigate the relationship between the functional correctness quality attribute (in terms of fault proneness and fault density) and the proposed coding standard violations-based measures. The results of such analysis are presented in Tables B.1 through B12 in the appendix B. in addition to the correlation significance and the impact direction of independent variables presented in Section 7.1.3, the results' characteristics will be addressed based on the following:

- Goodness of fit: evaluates the built model's performance when applied to the data set it was derived from. The obtained -2 log likelihood, Cox & snell R^2, and Nagelkerke R^2 are used with the logistic regression models and R^2 and adjusted R^2 with the linear regression models.
- Prediction accuracy: evaluates the built model's performance when applied to other datasets instead of the data set it was derived from. The prediction accuracy is explained in terms of the correct classification rate and receiver operating characteristic (ROC) curve for logistic regression models and mean absolute error (MAE) and root mean squared error (RMSE) for linear regression models.

7.2.1 Univariate Analysis With Respect To Fault Proneness

The results characteristics are discussed based on the goodness of fit and prediction accuracy. Regarding the goodness of fit, Tables 7.14 and 7.15 present the -2log likelihood for all systems under study. Since the lower values of -2log likelihood indicate a better fit than the higher values, the metrics are ordered according to the values of -2 log likelihood in increasing order to show which metrics' models are fitting better. In addition to -2log likelihood, another two measures are used, Cox & Snell R^2 and Nagelkerke R^2. The results for these two measures are presented in Tables 7.16 and 7.17 for all systems under study. Since the higher values of these two measures indicate a better fit than the lower values, the metrics are ordered accordingly in decreasing order to show which metrics' models are fitting better. In general, it can be observed that almost all -2log likelihood values are large. Furthermore, it can be observed that almost all Cox & Snell R^2 and Nagelkerke R^2 are relatively small for all models in the target set of systems except for few models in Synapse, Poi and Ant systems (those metrics' models at the beginning of Tables 7.16 and 7.17). Generally, this observation is reasonable due to the concentration of fault-proneness distribution presented in Table 6.1 which in turn implies that the probability estimates are also concentrated. The obtained values of these three measures imply that the models have low goodness of fits.

Table 7.14: Goodness of fit for Synapse, Velocity, Poi using -2Log likelihood

Synapse		Velocity		Poi	
Metric	-2LL	Metric	-2LL	Metric	-2LL
PSRV	281.532	PSCV	277.703	PSCV	452.52
PSCV	285.076	PSCVD	279.27	PSRV	456.642
PFCRV	291.806	PSRV	279.281	PExpCRV	475.096
PExpCRV	307.766	PComCRV	279.631	PDSCRV	498.078
PNCRV	310.764	PSRVD	279.718	PNCRV	525.379
PNCRVD	311.259	PNCRV	280.365	PPCICRV	549.122
PSCVD	311.429	PExpCRV	285.335	PFCRV	554.043
PSRVD	313.537	PFCRV	285.967	PExpCRVD	556.119
PComCRV	315.819	PExcCRV	287.24	PComCRVD	557.242
PPCICRV	315.993	PExcCRVD	287.24	PNCRVD	560.361
PComCRVD	316.807	PNCRVD	288.056	PSCVD	562.096
PFCRVD	317.996	PMCRV	291.02	PDSCRVD	564.253
PTCRVD	320.511	PDSCRVD	291.021	PTCRVD	566.684
PTCRV	320.882	PTCRV	292.176	PSRVD	567.201
PDSCRV	322.497	PPCICRV	292.511	PPCICRVD	570.112
PExpCRVD	323.004	PExpCRVD	293.118	PExcCRV	570.718
PMCRVD	323.606	PTCRVD	293.195	PExcCRVD	570.718
PMCRV	323.898	PFCRVD	293.212	PComCRV	571.598
PExcCRVD	324.709	PDSCRV	293.574	PMCRVD	572.375
PExcCRV	325.431	PMCRVD	293.696	PFCRVD	572.412
PPCICRVD	326.337	PPCICRVD	293.756	PMCRV	572.412
PDSCRVD	326.732	PComCRVD	293.766	PTCRV	572.485
PConCRV		PConCRV		PConCRV	
PConCRVD		PConCRVD		PConCRVD	

Table 7.15: Goodness of fit for Xalan, Camel, Ant using -2Log likelihood

Xalan		Camel		Ant		All Systems	
Metric	-2LL	Metric	-2LL	Metric	-2LL	Metric	-2LL
PSCV	1125.458	PSRV	898.055	PSRV	627.267	PSCV	3942.356
PSRV	1136.373	PSCV	898.318	PSCV	638.888	PSRV	3992.171
PSRVD	1137.865	PExpCRV	909.03	PExpCRV	683.86	PExpCRV	4187.015
PSCVD	1138.379	PFCRV	917.366	PNCRV	699.161	PDSCRV	4208.04
PComCRVD	1155.213	PNCRV	918.6	PFCRV	700.331	PPCICRV	4227.479
PDSCRV	1155.898	PComCRV	919.44	PComCRV	721.045	PNCRV	4330.479
PPCICRV	1160.694	PFCRVD	926.17	PSCVD	728.669	PSCVD	4373.708
PNCRV	1166.477	PExpCRVD	928.158	PMCRV	734.717	PFCRV	4381.469
PNCRVD	1172.902	PMCRV	929.025	PSRVD	743.702	PSRVD	4392.424
PExpCRV	1178.589	PPCICRV	930.405	PTCRV	745.441	PMCRV	4406.389
PMCRV	1191.729	PExcCRV	931.64	PPCICRV	752.547	PNCRVD	4411.415
PComCRV	1192.331	PExcCRVD	931.64	PDSCRV	756.907	PComCRVD	4421.315
PFCRV	1198.573	PComCRVD	932.381	PNCRVD	757.021	PComCRV	4421.884
PExpCRVD	1198.872	PDSCRV	933.38	PComCRVD	771.762	PTCRV	4446.317
PTCRV	1200.047	PMCRVD	933.534	PExcCRV	774.055	PTCRVD	4446.317
PFCRVD	1200.597	PDSCRVD	933.735	PDSCRVD	782.55	PPCICRVD	4473.834
PDSCRVD	1201.765	PTCRVD	934.032	PFCRVD	785.294	PExcCRV	4482.136
PTCRVD	1203.921	PSRVD	934.296	PMCRVD	786.478	PMCRVD	4486.232
PConCRV	1206.768	PNCRVD	934.566	PExcCRVD	787.771	PFCRVD	4486.491
PConCRVD	1206.768	PSCVD	934.669	PTCRVD	787.826	PExcCRVD	4487.172
PExcCRV	1207.42	PPCICRVD	934.822	PConCRV	787.844	PConCRVD	4487.505
PExcCRVD	1208.651	PTCRV	934.833	PConCRVD	787.844	PConCRV	4487.742
PPCICRVD	1209.233	PConCRV		PPCICRVD	787.911	PExpCRVD	4488.489
PMCRVD	1209.765	PConCRVD		PExpCRVD	788.186	PDSCRVD	4488.929

Table 7.16: Goodness of fit for synapse, Velocity, Poi using Cox&snell R^2 and Nagelkerke R^2

Synapse			Velocity			Poi		
Metric	Cox & snell R^2	Nagelkerke R^2	Metric	Cox & snell R^2	Nagelkerke R^2	Metric	Cox & snell R^2	Nagelkerke R^2
PSRV	0.162	0.225	PSCV	0.068	0.094	PSCV	0.239	0.328
PSCV	0.150	0.209	PSCVD	0.061	0.085	PSRV	0.232	0.318
PFCRV	0.128	0.177	PSRV	0.061	0.085	PExpCRV	0.199	0.273
PExpCRV	0.072	0.099	PComCRV	0.060	0.083	PDSCRV	0.156	0.214
PNCRV	0.061	0.084	PSRVD	0.060	0.082	PNCRV	0.102	0.140
PNCRVD	0.059	0.082	PNCRV	0.057	0.079	PPCICRV	0.052	0.071
PSCVD	0.058	0.081	PExpCRV	0.036	0.050	PFCRV	0.041	0.056
PSRVD	0.051	0.070	PFCRV	0.034	0.046	PExpCRVD	0.037	0.050
PComCRV	0.042	0.058	PExcCRV	0.028	0.039	PComCRVD	0.034	0.047
PPCICRV	0.041	0.057	PExcCRVD	0.028	0.039	PNCRVD	0.027	0.037
PComCRVD	0.038	0.053	PNCRVD	0.025	0.034	PSCVD	0.023	0.032
PFCRVD	0.034	0.047	PDSCRVD	0.012	0.017	PDSCRVD	0.019	0.026
PTCRVD	0.024	0.034	PMCRV	0.012	0.017	PTCRVD	0.013	0.018
PTCRV	0.023	0.032	PTCRV	0.007	0.010	PSRVD	0.012	0.016
PDSCRV	0.017	0.023	PPCICRV	0.006	0.008	PPCICRVD	0.005	0.007
PExpCRVD	0.015	0.020	PExpCRVD	0.003	0.004	PExcCRV	0.004	0.006
PMCRVD	0.012	0.017	PTCRVD	0.003	0.004	PExcCRVD	0.004	0.006
PMCRV	0.011	0.016	PFCRVD	0.002	0.003	PComCRV	0.002	0.003
PExcCRVD	0.008	0.011	PDSCRV	0.001	0.001	PMCRVD	0.000	0.000
PExcCRV	0.005	0.007	PMCRVD	0.000	0.001	PFCRVD	0.000	0.000
PPCICRVD	0.002	0.003	PPCICRVD	0.000	0.000	PMCRV	0.000	0.000
PDSCRVD	0.000	0.000	PComCRVD	0.000	0.000	PTCRV	0.000	0.000
PConCRV			PConCRV			PConCRV		
PConCRVD			PConCRVD			PConCRVD		

Table 7.17: Goodness of fit for Xalan, Camel, Ant and all systems using Cox&snell R^2 and Nagelkerke R^2

Xalan			Camel			Ant			All Systems		
Metric	Cox & snell R^2	Nagel kerke R^2	Metric	Cox & snell R^2	Nagel kerke R^2	Metric	Cox & snell R^2	Nagel kerke R^2	Metric	Cox & snell R^2	Nagel kerke R^2
PSCV	0.092	0.123	PSRV	0.039	0.061	PSRV	0.195	0.298	PSCV	0.146	0.201
PSRV	0.080	0.107	PSCV	0.038	0.061	PSCV	0.183	0.279	PSRV	0.133	0.184
PSRVD	0.079	0.105	PExpCRV	0.027	0.043	PExpCRV	0.132	0.201	PExpCRV	0.083	0.115
PSCVD	0.078	0.105	PFCRV	0.019	0.029	PNCRV	0.113	0.173	PDSCRV	0.078	0.107
PComCRVD	0.060	0.081	PNCRV	0.017	0.027	PFCRV	0.112	0.171	PPCICRV	0.073	0.100
PDSCRV	0.060	0.080	PComCRV	0.016	0.026	PComCRV	0.087	0.133	PNCRV	0.045	0.062
PPCICRV	0.055	0.073	PFCRVD	0.009	0.015	PSCVD	0.077	0.118	PSCVD	0.033	0.045
PNCRV	0.048	0.064	PExpCRVD	0.007	0.011	PMCRV	0.070	0.107	PFCRV	0.030	0.042
PNCRVD	0.041	0.055	PMCRV	0.006	0.010	PSRVD	0.058	0.089	PSRVD	0.027	0.038
PExpCRV	0.035	0.047	PPCICRV	0.005	0.008	PTCRV	0.056	0.086	PMCRV	0.024	0.032
PMCRV	0.020	0.027	PExcCRV	0.003	0.005	PPCICRV	0.047	0.072	PNCRVD	0.022	0.030
PComCRV	0.020	0.026	PExcCRVD	0.003	0.005	PDSCRV	0.042	0.063	PComCRVD	0.019	0.027
PFCRV	0.013	0.017	PComCRVD	0.003	0.004	PNCRVD	0.041	0.063	PComCRV	0.019	0.026
PExpCRVD	0.012	0.017	PDSCRV	0.002	0.003	PComCRVD	0.022	0.034	PTCRV	0.012	0.017
PTCRV	0.011	0.015	PMCRVD	0.001	0.002	PExcCRV	0.019	0.029	PTCRVD	0.012	0.017
PFCRVD	0.010	0.014	PDSCRVD	0.001	0.002	PDSCRVD	0.008	0.012	PPCICRVD	0.004	0.006
PDSCRVD	0.009	0.012	PTCRVD	0.001	0.001	PFCRVD	0.004	0.006	PExcCRV	0.002	0.003
PTCRVD	0.007	0.009	PSRVD	0.001	0.001	PMCRVD	0.003	0.004	PMCRVD	0.001	0.001
PConCRVD	0.003	0.005	PNCRVD	0.000	0.000	PExcCRVD	0.001	0.001	PFCRVD	0.001	0.001
PConCRV	0.003	0.005	PSCVD	0.000	0.000	PTCRVD	0.001	0.001	PExcCRVD	0.001	0.001
PExcCRV	0.003	0.004	PPCICRVD	0.000	0.000	PConCRV	0.001	0.001	PConCRVD	0.000	0.001
PExcCRVD	0.001	0.002	PTCRV	0.000	0.000	PConCRVD	0.001	0.001	PConCRV	0.000	0.000
PPCICRVD	0.001	0.001	PConCRVD			PPCICRVD	0.001	0.001	PExpCRVD	0.000	0.000
PMCRVD	0.000	0.000	PConCRV			PExpCRVD	0.000	0.000	PDSCRVD	0.000	0.000

Regarding the model's prediction accuracy, the predictive accuracy of the prediction models is evaluated using the correct classification rate (accuracy) and receiver operating characteristic (ROC) curve. ROC curve area is a graphical plot which illustrates the performance of a binary classifier as its discrimination threshold is varied. Since the higher values of ROC are always better, the metrics' models are ordered accordingly in decreasing order to show which metrics' models are practical predictors for fault proneness. Tables 7.18 and 7.19 present the results of prediction accuracy for all systems under study. By adopting the categorization of ROC values presented in [46], it can be observed that most ROC values fall into the "fair" category which indicates that most built models are not practical classifiers for fault-prone and non-fault-prone classes. The exceptions only for PSRV, PSCV, PExpCRV and PDSCRV in Synapse, Poi and Ant systems. These metrics' models fall in "acceptable" category and marked in boldface in Tables 7.18 and 7.19. As stated by [54] "a measure might be found to be a statistically significant quality predictor (p-value<0.05), but it could be determined to be an impractical predictor according to the ROC". Many of the proposed measures were found to be significantly correlated with fault-proneness but they are determined as impractical predictors according to ROC. Tables B.1 through B.6 in the appendix B present the univariate logistic regression results for all systems under study.

Table 7.18: prediction accuracy for synapse, Velocity, Poi using correct classification rate and ROC

Synapse			Velocity			Poi		
Metric	Correct Classification	ROC	Metric	Correct Classification	ROC	Metric	Correct Classification	ROC
PSRV	69.5313	0.739	PSRV	64.1921	0.649	**PSRV**	74.0319	0.781
PSCV	71.4844	0.726	PSCV	63.7555	0.646	**PSCV**	73.3485	0.781
PSCVD	64.8438	0.685	PSRVD	61.1354	0.638	**PExpCRV**	76.082	0.715
PSRVD	65.2344	0.671	PSCVD	62.0087	0.634	**PDSCRV**	75.1708	0.704
PFCRV	69.9219	0.665	PNCRV	63.3188	0.585	PExpCRVD	64.0091	0.666
PFCRVD	66.7969	0.637	PComCRV	65.9389	0.57	PDSCRVD	64.0091	0.647
PNCRVD	66.4063	0.632	PFCRV	66.8122	0.561	PNCRV	69.7039	0.638
PExpCRV	68.3594	0.632	PExpCRV	64.1921	0.553	PSCVD	64.9203	0.575
PNCRV	69.1406	0.612	PNCRVD	65.9389	0.527	PPCICRV	64.0091	0.574
PExpCRVD	65.2344	0.6	PExcCRV	67.2489	0.507	PFCRV	64.0091	0.563
PComCRV	66.4063	0.558	PExcCRVD	67.2489	0.507	PSRVD	63.7813	0.55
PComCRVD	66.4063	0.553	PTCRVD	65.5022	0.506	PNCRVD	65.6036	0.549
PPCICRV	68.3594	0.537	PExpCRVD	65.5022	0.505	PComCRVD	68.3371	0.526
PDSCRV	65.625	0.522	PDSCRVD	65.9389	0.504	PMCRV	64.0091	0.504
PDSCRVD	66.4063	0.518	PTCRV	64.1921	0.503	PMCRVD	64.0091	0.499
PTCRV	67.9688	0.504	PMCRV	65.5022	0.501	PTCRVD	64.6925	0.493
PTCRVD	67.1875	0.504	PPCICRV	65.9389	0.498	PExcCRV	64.0091	0.491
PMCRV	67.1875	0.495	PMCRVD	65.5022	0.477	PExcCRVD	64.0091	0.491
PMCRVD	66.7969	0.495	PPCICRVD	65.9389	0.444	PFCRVD	64.0091	0.481
PExcCRV	66.4063	0.486	PDSCRV	65.9389	0.436	PComCRV	64.0091	0.481
PExcCRVD	66.7969	0.486	PFCRVD	65.9389	0.42	PTCRV	63.7813	0.466
PPCICRVD	66.4063	0.419	PComCRVD	65.5022	0.417	PPCICRVD	64.0091	0.461
PConCRV			PConCRV			PConCRV		
PConCRVD			PConCRVD			PConCRVD		

Table 7.19: prediction accuracy for Xalan, Camel, Ant and all systems using correct classification rate (CCR) and ROC

Xalan			Camel			Ant			All Systems		
Metric	CCR	ROC	Metric	CCR	ROC	Metric	CCR	ROC	Metric	CCR	ROC
PSRVD	64.229	0.666	PSCV	79.4212	0.631	**PSRV**	81.2416	0.793	**PSCV**	72.4158	0.726
PSCVD	62.514	0.665	PSRV	79.8499	0.63	**PSCV**	80.4318	0.779	**PSRV**	70.1699	0.72
PSCV	65.486	0.664	PNCRV	79.9571	0.577	**PExpCRV**	78.9474	0.717	PExpCRV	67.2329	0.661
PSRV	60.229	0.66	PExpCRVD	79.8499	0.559	PFCRV	76.1134	0.699	PDSCRV	67.2617	0.642
PDSCRV	58.629	0.605	PFCRVD	80.1715	0.555	PSCVD	77.5978	0.69	PSCVD	65.1886	0.613
PExpCRV	56.688	0.594	PExpCRV	80.2787	0.555	PNCRV	77.5978	0.682	PNCRV	62.482	0.607
PNCRVD	61.143	0.588	PComCRV	79.9571	0.552	PSRVD	77.5978	0.67	PSRVD	65.1886	0.605
PPCICRV	62.171	0.572	PFCRV	79.8499	0.548	PComCRV	77.5978	0.624	PExpCRVD	65.1598	0.605
PNCRV	54.4	0.566	PComCRVD	79.9571	0.531	PNCRVD	77.5978	0.617	PPCICRV	69.3637	0.599
PComCRVD	58.171	0.558	PDSCRV	79.9571	0.5	PTCRV	77.4629	0.594	PPCICRVD	64.8719	0.593
PFCRV	55.886	0.543	PPCICRV	80.0643	0.498	PDSCRV	78.1377	0.576	PFCRV	65.5629	0.58
PComCRV	55.427	0.531	PMCRVD	79.8499	0.498	PMCRVD	77.3279	0.573	PComCRV	65.1886	0.558
PMCRV	56.8	0.529	PMCRV	80.0643	0.497	PMCRV	79.4872	0.564	PNCRVD	65.1886	0.542
PPCICRVD	51.543	0.527	PExcCRV	79.9571	0.493	PPCICRV	77.8677	0.561	PMCRVD	65.0446	0.536
PTCRV	55.771	0.518	PExcCRVD	79.9571	0.493	PPCICRVD	77.5978	0.56	PMCRV	66.7434	0.532
PExcCRV	53.029	0.503	PNCRVD	79.9571	0.488	PComCRVD	77.5978	0.528	PComCRVD	65.1886	0.527
PExcCRVD	53.029	0.503	PTCRVD	79.8499	0.487	PExcCRVD	77.4629	0.518	PTCRV	64.9295	0.525
PConCRV	53.257	0.497	PTCRV	79.9571	0.481	PExcCRV	78.2726	0.511	PExcCRVD	65.1598	0.506
PConCRVD	53.257	0.497	PPCICRVD	79.9571	0.471	PConCRV	77.5978	0.491	PExcCRV	65.2462	0.504
PMCRVD	52.686	0.495	PDSCRVD	79.9571	0.47	PConCRVD	77.5978	0.491	PConCRVD	65.1598	0.5
PFCRVD	52.114	0.485	PSRVD	79.9571	0.46	PFCRVD	77.5978	0.426	PConCRV	65.2174	0.499
PTCRVD	53.029	0.462	PSCVD	79.9571	0.441	PDSCRVD	77.5978	0.411	PTCRVD	65.1886	0.464
PDSCRVD	53.029	0.461	PConCRV			PExpCRVD	77.5978	0.389	PFCRVD	65.1886	0.448
PExpCRVD	51.657	0.455	PConCRVD			PTCRVD	77.5978	0.378	PDSCRVD	65.1886	0.434

7.2.2 Univariate Analysis With Respect To Fault Density

The results characteristics will also be discussed based on the goodness of fit and the prediction accuracy. Regarding the goodness of fit, Tables 7.20 and 7.21 present the R^2 and adjusted R^2 for all systems under study. R^2 and adjusted R^2 tell us what percentage of the variability in the dependent variable can be explained by the predictors in the model. It is often that adjusted R^2 smaller than R^2. Since the higher values of these two measures indicate a better fit than the lower values, the metrics are ordered accordingly in decreasing order to show which linear regression models are fitting better. The results for these two measures are presented in Tables 7.20 and 7.21 for all systems under study. In general, the goodness of fit results show that almost all R^2 and adjusted R^2 values are relatively small in the target set of systems which implies that the models have low goodness of fits. The results presented in Tables 7.20 and 7.21 confirm what stated by [54]: "for technical reasons, high values of R^2 are rare, even for accurate models". Furthermore, such observation might be due to the concentration of faults distribution as presented in Table 6.1.

Table 7.20: Goodness of fit for synapse, Velocity, Poi using R^2 and adjusted R^2

Synapse			Velocity			Poi		
Metric	R^2	Adj. R^2	Metric	R^2	Adj. R^2	Metric	R^2	Adj. R^2
PTCRVD	0.350	0.347	PDSCRV	0.012	0.008	PSRVD	0.040	0.038
PFCRVD	0.111	0.107	PFCRV	0.010	0.005	PSCVD	0.039	0.036
PTCRV	0.083	0.079	PExpCRV	0.007	0.003	PDSCRVD	0.026	0.023
PSRVD	0.051	0.047	PSRV	0.007	0.003	PComCRV	0.017	0.015
PSCVD	0.049	0.045	PTCRVD	0.005	0.001	PNCRVD	0.010	0.008
PExpCRVD	0.048	0.045	PPCICRV	0.004	0.000	PPCICRV	0.008	0.005
PPCICRVD	0.017	0.013	PSCV	0.004	0.000	PComCRVD	0.006	0.003
PComCRV	0.010	0.006	PMCRV	0.004	-0.001	PExpCRVD	0.005	0.002
PSCV	0.008	0.004	PComCRV	0.003	-0.001	PSCV	0.003	0.001
PPCICRV	0.007	0.003	PPCICRVD	0.003	-0.001	PMCRV	0.003	0.000
PNCRVD	0.004	0.001	PNCRVD	0.003	-0.001	PMCRVD	0.003	0.000
PFCRV	0.004	0.001	PFCRVD	0.002	-0.002	PFCRV	0.002	0.000
PMCRVD	0.003	-0.001	PTCRV	0.002	-0.002	PTCRV	0.002	0.000
PSRV	0.003	-0.001	PExpCRVD	0.002	-0.002	PSRV	0.002	0.000
PDSCRV	0.002	-0.002	PSCVD	0.002	-0.002	PFCRVD	0.002	0.000
PComCRVD	0.002	-0.002	PComCRVD	.001	-.004	PNCRV	0.002	0.000
PExcCRV	0.000	-0.003	PDSCRVD	0.001	-0.004	PPCICRVD	0.001	-0.001
PNCRV	0.000	-0.004	PMCRVD	0.001	-0.004	PTCRVD	0.001	-0.001
PExcCRVD	0.000	-0.004	PSRVD	0.001	-0.004	PExpCRV	0.000	-0.002
PMCRV	0.000	-0.004	PExcCRV	0.000	-0.004	PDSCRV	0.000	-0.002
PExpCRV	0.000	-0.004	PNCRV	0.000	-0.004	PExcCRVD	0.000	-0.002
PDSCRVD	0.000	-0.004	PExcCRVD	0.000	-0.004	PExcCRV	0.000	-0.002
PConCRV			PConCRV			PConCRV		
PConCRVD			PConCRVD			PConCRVD		

Table 7.21: Goodness of fit for Xalan, Camel, Ant and all systems using R^2 and adjusted R^2

Xalan			Camel			Ant			All systems		
Metric	R^2	Adj. R^2	Metric	R^2	Adj. R^2	Metric	R^2	Adj. R^2	Metric	R^2	Adj. R^2
PComCRV	0.045	0.044	PSCVD	0.026	0.025	PSRV	0.008	0.006	PSCVD	0.012	0.011
PSCV	0.030	0.029	PSRVD	0.023	0.022	PNCRV	0.008	0.007	PSRVD	0.011	0.010
PSRV	0.028	0.027	PComCRVD	0.008	0.007	PExpCRV	0.006	0.005	PComCRV	0.009	0.009
PNCRVD	0.024	0.023	PFCRVD	0.006	0.005	PSCV	0.006	0.005	PNCRVD	0.005	0.004
PExpCRV	0.020	0.019	PExcCRV	0.003	0.002	PMCRV	0.005	0.004	PSRV	0.003	0.003
PFCRV	0.019	0.018	PExcCRVD	0.003	0.002	PFCRVD	0.003	0.002	PFCRV	0.003	0.003
PDSCRV	0.018	0.017	PNCRVD	0.002	0.001	PExcCRV	0.003	0.002	PExpCRV	0.003	0.003
PSCVD	0.012	0.011	PComCRV	0.002	0.001	PComCRV	0.003	0.002	PSCV	0.003	0.002
PSRVD	0.010	0.009	PDSCRV	0.001	0.000	PDSCRV	0.002	0.001	PMCRV	0.002	0.001
PPCICRV	0.006	0.005	PNCRV	0.001	0.000	PNCRVD	0.001	0	PDSCRV	0.002	0.001
PMCRV	0.006	0.005	PSRV	0.001	0.000	PPCICRV	0.001	0	PFCRVD	0.001	0.001
PComCRVD	0.005	0.004	PExpCRV	0.001	0.000	PFCRV	0.001	0	PTCRV	0.001	0.001
PFCRVD	0.005	0.004	PDSCRVD	0.001	0.000	PMCRVD	0.001	-0.001	PNCRV	0.001	0.001
PTCRV	0.004	0.002	PMCRV	0.001	0.000	PDSCRVD	0.001	0	PPCICRVD	0.001	0.000
PNCRV	0.003	0.001	PMCRVD	0.001	0.000	PExcCRVD	0.001	0	PTCRVD	0.000	0.000
PExcCRV	0.003	0.001	PSCV	0.001	0.000	PTCRV	0.001	0	PMCRVD	0.000	0.000
PTCRVD	0.001	0.000	PPCICRV	0.000	-0.001	PTCRVD	0.001	0	PPCICRV	0.000	0.000
PPCICRVD	0.001	0.000	PPCICRVD	0.000	-0.001	PSCVD	0.001	-0.001	PExcCRV	0.000	0.000
PExpCRVD	0.001	-0.001	PTCRVD	0.000	-0.001	PSRVD	0.000	-0.001	PComCRVD	0.000	0.000
PExcCRVD	0.000	-0.001	PExpCRVD	0.000	-0.001	PPCICRVD	0.000	-0.001	PConCRVD	0.000	0.000
PDSCRVD	0.000	-0.001	PFCRV	0.000	-0.001	PExpCRVD	0.000	-0.001	PExcCRVD	0.000	0.000
PMCRVD	0.000	-0.001	PTCRV	0.000	-0.001	PConCRV	0.000	-0.001	PDSCRVD	0.000	0.000
PConCRV			PConCRV			PConCRVD	0.000	-0.001	PExpCRVD	0.000	0.000
PConCRVD			PConCRVD			PComCRVD	0.000	-0.001	PConCRV	0.000	0.000

Regarding the model's prediction accuracy, the predictive accuracy of the prediction models is evaluated using the mean absolute error (MAE) and the root mean squared error (RMSE). All these measures are based on what so called residual which is, the difference between the predicted and the observed values. The results of the prediction accuracy are analyzed in terms of these two measures. The lower values of these two measures are always better than the higher values. Additionally, the values of RMSE are always higher than MAE. So the metrics' models are ordered according to the MAE in increasing order to show which metrics' models more accurate predictors for fault density. Tables 7.22 and 7.23 present the ordered results of the prediction accuracy for all linear regression models in all case studies investigated by this study in addition to the case study that comprises all observations from all systems (the last four columns in Table 7.23).

It can be observed in Tables 7.22 and 7.23 that the best accuracy results of the linear regression models were achieved in Ant system while the worst accuracy results were achieved in Camel system. Table 7.24 presents the average accuracy results in terms of MAE for each linear regression model in all case studies. It can be observed that all regression models achieved closed accuracy results. Furthermore, Table 7.25 presents the average accuracy results in terms of RMSE for each linear regression model in all case studies. It can also be observed that all regression models achieved also closed accuracy results based on this measure. Tables B.7 through B.12 in the appendix B present the univariate linear regression results for all systems under study.

Table 7.22: The accuracy of metrics models using MAE, RMSE and AE Std for Synapse, Velocity, Poi Systems

Synapse				Velocity				Poi			
Metric	MAE	RMSE	AE Std	Metric	MAE	RMSE	AE Std	Metric	MAE	RMSE	AE Std
PFCRVD	11.2783	22.6904	19.73	PDSCRV	17.066	29.328	23.85	PDSCRVD	12.560	22.847	19.08
PTCRVD	11.7584	21.8449	18.45	PMCRV	17.080	29.416	23.95	PSCVD	12.581	22.676	18.86
PExpCRVD	12.14	23.3557	19.99	PNCRV	17.175	29.571	24.07	PSRVD	12.635	22.619	18.76
PSCV	12.3084	23.0214	19.49	PExcCRVD	17.177	29.472	23.95	PExpCRVD	12.898	22.922	18.95
PFCRV	12.3416	22.9951	19.44	PFCRV	17.182	29.335	23.78	PComCRV	13.125	22.768	18.60
PSRV	12.4005	23.0089	19.42	PFCRVD	17.183	29.452	23.92	PNCRVD	13.150	22.886	18.73
PPCICRV	12.4581	22.9582	19.32	PTCRV	17.212	29.539	24.01	PMCRV	13.171	22.882	18.71
PTCRV	12.4825	23.4925	19.94	PExcCRV	17.216	29.477	23.93	PFCRV	13.176	22.947	18.79
PExcCRV	12.5449	22.9309	19.23	PComCRVD	17.226	29.610	24.08	PMCRVD	13.179	22.884	18.71
PComCRVD	12.5474	22.9573	19.26	PTCRVD	17.255	29.705	24.18	PTCRVD	13.228	22.913	18.71
PExpCRV	12.5543	22.9648	19.27	PPCICRV	17.269	29.423	23.82	PDSCRV	13.240	23.003	18.81
PDSCRVD	12.573	22.9942	19.29	PPCICRVD	17.269	29.423	23.82	PTCRV	13.247	22.974	18.77
PDSCRV	12.5758	22.9438	19.23	PDSCRVD	17.269	29.658	24.11	PComCRVD	13.265	22.967	18.75
PNCRV	12.5759	22.9743	19.26	PMCRVD	17.270	29.572	24.01	PExcCRV	13.284	22.940	18.70
PExcCRVD	12.5803	22.9461	19.23	PExpCRVD	17.296	29.512	23.91	PExpCRV	13.319	23.002	18.75
PMCRV	12.5839	22.9812	19.27	PSRVD	17.296	29.549	23.96	PNCRV	13.329	22.995	18.74
PMCRVD	12.6041	23.0793	19.37	PNCRVD	17.300	29.558	23.97	PPCICRV	13.329	22.850	18.56
PPCICRVD	12.7534	23.4274	19.69	PSCV	17.308	29.451	23.83	PExcCRVD	13.331	23.029	18.78
PNCRVD	12.763	23.0627	19.26	PSRV	17.340	29.420	23.77	PSRV	13.344	23.008	18.74
PComCRV	12.8916	23.0227	19.11	PSCVD	17.342	29.541	23.92	PFCRVD	13.353	23.021	18.75
PSRVD	13.3093	23.2923	19.15	PExpCRV	17.385	29.423	23.74	PPCICRVD	13.357	22.979	18.70
PSCVD	13.3207	23.2369	19.08	PComCRV	17.385	29.521	23.86	PSCV	13.406	22.994	18.68
PConCRV				PConCRV				PConCRV			
PConCRVD				PConCRVD				PConCRVD			

Table 7.23: The accuracy of metrics models using MAE, RMSE and AE Std for Xalan, Camel, Ant and All systems

Xalan				Camel				Ant				All Systems			
Metric	MAE	RMSE	AE Std	Metric	MAE	RMSE	AE Std	Metric	MAE	RMSE	AE Std	Metric	MAE	RMSE	AE Std
PComCRV	10.400	19.151	16.08	PSRV	19.309	43.206	38.67	PSRV	4.462	9.512	8.41	PPCICRVD	12.784	28.145	25.08
PPCICRV	10.469	19.513	16.46	PSRVD	18.885	42.856	38.49	PSCV	4.494	9.519	8.40	PFCRVD	12.806	28.138	25.06
PExpCRV	10.493	19.378	16.29	PNCRV	19.273	43.192	38.67	PNCRV	4.521	9.510	8.37	PSRVD	12.813	28.047	24.95
PNCRVD	10.500	19.357	16.26	PNCRVD	19.111	43.227	38.79	PExpCRV	4.538	9.520	8.37	PDSCRVD	12.821	28.138	25.05
PExcCRV	10.514	19.543	16.47	PPCICRV	19.151	43.190	38.73	PMCRV	4.612	9.521	8.33	PComCRVD	12.821	28.148	25.06
PComCRVD	10.521	19.519	16.44	PPCICRVD	19.135	43.191	38.74	PComCRV	4.615	9.536	8.35	PExcCRV	12.823	28.134	25.05
PMCRV	10.533	19.510	16.42	PFCRV	19.213	43.231	38.75	PFCRV	4.643	9.545	8.34	PConCRV	12.824	28.134	25.04
PPCICRVD	10.540	19.587	16.51	PFCRVD	18.895	43.204	38.87	PTCRV	4.658	9.537	8.33	PExcCRVD	12.825	28.138	25.05
PExpCRVD	10.541	19.557	16.47	PMCRV	19.188	43.170	38.69	PDSCRV	4.664	9.541	8.33	PConCRVD	12.826	28.136	25.05
PTCRVD	10.544	19.589	16.51	PMCRVD	19.166	43.173	38.71	PPCICRV	4.665	9.540	8.33	PTCRV	12.827	28.122	25.03
PExcCRVD	10.547	19.571	16.48	PDSCRV	19.209	43.181	38.69	PExcCRV	4.669	9.539	8.32	PTCRVD	12.829	28.144	25.05
PConCRV	10.548	19.565	16.48	PDSCRVD	19.129	43.182	38.74	PSCVD	4.677	9.562	8.35	PMCRVD	12.829	28.133	25.04
PDSCRVD	10.554	19.591	16.50	PExpCRV	19.266	43.197	38.68	PSRVD	4.694	9.562	8.34	PExpCRVD	12.831	28.140	25.05
PTCRV	10.554	19.533	16.43	PExpCRVD	19.186	43.212	38.74	PDSCRVD	4.705	9.536	8.30	PSCVD	12.837	28.047	24.94
PSRVD	10.561	19.653	16.57	PExcCRV	19.077	43.184	38.76	PFCRVD	4.707	9.533	8.30	PMCRV	12.841	28.112	25.01
PConCRVD	10.564	19.583	16.49	PExcCRVD	19.077	43.184	38.76	PComCRVD	4.709	9.551	8.31	PNCRVD	12.841	28.082	24.98
PFCRVD	10.565	19.516	16.41	PTCRV	19.145	43.191	38.74	PTCRVD	4.713	9.536	8.30	PPCICRV	12.849	28.141	25.04
PSCVD	10.569	19.756	16.69	PTCRVD	19.172	43.237	38.77	PConCRV	4.716	9.539	3.24	PFCRV	12.872	28.105	24.99
PMCRVD	10.571	19.598	16.50	PComCRV	19.277	43.209	38.69	PConCRVD	4.716	9.539	8.30	PComCRV	12.879	28.021	24.89
PNCRV	10.579	19.560	16.45	PComCRVD	18.981	43.226	38.86	PPCICRVD	4.720	9.542	8.30	PDSCRV	12.897	28.121	24.99
PSCV	10.586	19.296	16.13	PSCV	19.277	43.221	38.71	PExpCRVD	4.721	9.553	8.31	PNCRV	12.900	28.133	25.00
PFCRV	10.612	19.394	16.23	PSCVD	18.930	42.829	38.44	PExcCRVD	4.726	9.584	8.34	PExpCRV	12.939	28.106	24.95
PDSCRV	10.613	19.409	16.25	PConCRV				PMCRVD	4.729	9.563	8.32	PSRV	12.981	28.104	24.93
PSRV	10.655	19.307	16.10	PConCRVD				PNCRVD	4.764	9.596	8.34	PSCV	12.987	28.114	24.94

Table 7.24: The prediction accuracy average for linear regression models using MAE

Mean Absolute Error (MAE)								
Metric	Synapse	Velocity	Poi	Xalan	Camel	Ant	All systems	Avg.
PSRV	12.4005	17.340	13.344	10.655	19.309	4.462	12.981	12.93
PSRVD	13.3093	17.296	12.635	10.561	18.885	4.694	12.813	12.88
PNCRV	12.5759	17.175	13.329	10.579	19.273	4.521	12.900	12.91
PNCRVD	12.763	17.300	13.150	10.500	19.111	4.764	12.841	12.92
PPCICRV	12.4581	17.269	13.329	10.469	19.151	4.665	12.849	12.88
PPCICRVD	12.7534	17.269	13.357	10.540	19.135	4.720	12.784	12.94
PFCRV	12.3416	17.182	13.176	10.612	19.213	4.643	12.872	12.86
PFCRVD	11.2783	17.183	13.353	10.565	18.895	4.707	12.806	12.68
PMCRV	12.5839	17.080	13.171	10.533	19.188	4.612	12.841	12.86
PMCRVD	12.6041	17.270	13.179	10.571	19.166	4.729	12.829	12.91
PDSCRV	12.5758	17.066	13.240	10.613	19.209	4.664	12.897	12.89
PDSCRVD	12.573	17.269	12.560	10.554	19.129	4.705	12.821	12.80
PExpCRV	12.5543	17.385	13.319	10.493	19.266	4.538	12.939	12.93
PExpCRVD	12.14	17.296	12.898	10.541	19.186	4.721	12.831	12.80
PExcCRV	12.5449	17.216	13.284	10.514	19.077	4.669	12.823	12.88
PExcCRVD	12.5803	17.177	13.331	10.547	19.077	4.726	12.825	12.89
PTCRV	12.4825	17.212	13.247	10.554	19.145	4.658	12.827	12.88
PTCRVD	11.7584	17.255	13.228	10.544	19.172	4.713	12.829	12.79
PConCRV				10.548		4.716	12.824	
PConCRVD				10.564		4.716	12.826	
PComCRV	12.8916	17.385	13.125	10.400	19.277	4.615	12.879	12.94
PComCRVD	12.5474	17.226	13.265	10.521	18.981	4.709	12.821	12.87
PSCV	12.3084	17.308	13.406	10.586	19.277	4.494	12.987	12.91
PSCVD	13.3207	17.342	12.581	10.569	18.930	4.677	12.837	12.89
Number of Observations	**256**	**229**	**439**	**875**	**933**	**741**	**3473**	

Table 7.25: The prediction accuracy average for linear regression models using RMSE

Root Mean Squared Error (RMSE)								
Metric	Synapse	Velocity	Poi	Xalan	Camel	Ant	All systems	Avg.
PSRV	23.0089	29.420	23.008	19.307	43.206	9.512	28.104	25.08
PSRVD	23.2923	29.549	22.619	19.653	42.856	9.562	28.047	25.08
PNCRV	22.9743	29.571	22.995	19.560	43.192	9.510	28.133	25.13
PNCRVD	23.0627	29.558	22.886	19.357	43.227	9.596	28.082	25.11
PPCICRV	22.9582	29.423	22.850	19.513	43.190	9.540	28.141	25.09
PPCICRVD	23.4274	29.423	22.979	19.587	43.191	9.542	28.145	25.18
PFCRV	22.9951	29.335	22.947	19.394	43.231	9.545	28.105	25.08
PFCRVD	22.6904	29.452	23.021	19.516	43.204	9.533	28.138	25.08
PMCRV	22.9812	29.416	22.882	19.510	43.170	9.521	28.112	25.08
PMCRVD	23.0793	29.572	22.884	19.598	43.173	9.563	28.133	25.14
PDSCRV	22.9438	29.328	23.003	19.409	43.181	9.541	28.121	25.08
PDSCRVD	22.9942	29.658	22.847	19.591	43.182	9.536	28.138	25.14
PExpCRV	22.9648	29.423	23.002	19.378	43.197	9.520	28.106	25.08
PExpCRVD	23.3557	29.512	22.922	19.557	43.212	9.553	28.140	25.18
PExcCRV	22.9309	29.477	22.940	19.543	43.184	9.539	28.134	25.11
PExcCRVD	22.9461	29.472	23.029	19.571	43.184	9.584	28.138	25.13
PTCRV	23.4925	29.539	22.974	19.533	43.191	9.537	28.122	25.20
PTCRVD	21.8449	29.705	22.913	19.589	43.237	9.536	28.144	25.00
PConCRV				19.565		9.539	28.134	
PConCRVD				19.583		9.539	28.136	
PComCRV	23.0227	29.521	22.768	19.151	43.209	9.536	28.021	25.03
PComCRVD	22.9573	29.610	22.967	19.519	43.226	9.551	28.148	25.14
PSCV	23.0214	29.451	22.994	19.296	43.221	9.519	28.114	25.09
PSCVD	23.2369	29.541	22.676	19.756	42.829	9.562	28.047	25.09
Number of Observations	**256**	**229**	**439**	**875**	**933**	**741**	**3473**	

7.3 Multivariate Analysis

For each one of the target set of systems under study, two types of multivariate regression models are built. One of them for the fault proneness dependent variable while the other for the fault density dependent variable. Each one of those models comprises a set of variables (selected attributes) from the independent variables sets which in our case are: (1) the coding standard violations-based metrics, (2) CK metrics and (3) a combination of the two sets of metrics. In other words, the models are built in three different ways:

- Using subset of both coding standard violations-based metrics and CK metrics as independent variables.
- Using only subset of coding standard violations-based metrics as independent variables.
- Using only subset of CK metrics as independent variables.

Each subset for each system under study was selected according to the attribute selection procedure. In addition to the correlation significance and the impact direction of independent variables presented in section 7.1.3, the results' characteristics are addressed based on following:

- Goodness of fit: evaluates the built model performance when applied to the data set it was derived from. The obtained -2 log likelihood, Cox & snell R^2, and Nagelkerke R^2 are used with the logistic regression models and R^2 and adjusted R^2 are used with the linear regression models.
- Prediction accuracy: evaluates the built model performance when applied to other datasets instead of the data set it was derived from. The prediction accuracy is explained in terms of the correct classification rate and receiver operating characteristic (ROC) curve for logistic regression models and mean absolute error (MAE) for linear regression models.

7.3.1 Attribute Selection With Respect To Fault Proneness

Results of correlation and PCA show that some of the metrics capture similar dimensions in the data set which reflecting the fact that some of them contain redundant or noisy information. Therefore, to build accurate models to predict which classes are

faults prone, attribute selection procedure is performed as described in Section 6.3.4. The results of the attribute selection against the fault-proneness, as dependent variable for all systems under study, are presented in Tables 7.26, 7.27 and 7.28.

As shown in these tables, some metrics are selected more than one time as fault-proneness predictors across systems under study, some of them are selected only once, and some others have never been selected in all systems under study. The coding standard violations-based metrics that have never been selected, at any system from the target set of this study, as predictors for the fault-proneness of the class, are PSRV, PFCRV, PMCRV, PExcCRV, PExcCRVD, PTCRV, PConCRV, PConCRVD, PComCRV. The reason could be the high correlation of these metrics with some other metrics as presented in appendix E, Tables E.1 through E.6. Another reason could be that, in the systems investigated in this research, these metrics might not be useful indicators of the fault-proneness of the class. Thus, they are removed from the logistic regression models as presented in the appendix D. However, it cannot be generalized that these metrics are not useful predictors for the fault-proneness of the classes in other case studies.

Regarding the CK metrics, the metrics that have never been selected as predictors for the fault-proneness of the class in all systems under study are NOC and DIT. So these metrics are removed from the logistic regression models as presented in the appendix D.

Table 7.26: Selected attributes from CK suite with fault-proneness for all systems

CK Metrics						
	WMC	DIT	NOC	CBO	RFC	LCOM
Synapse	√			√	√	√
Velocity	√			√	√	√
Poi				√	√	√
Xalan	√			√	√	√
Camel	√			√	√	√
Ant	√			√	√	√

Table 7.27: Selected attributes from coding standard violations-based suite with fault-proneness for all systems

Coding Standard Violations-Based Metrics																								
	PSRV	PSRVD	PNCRV	PNCRVD	PPCICRV	PPCICRVD	PFCRV	PFCRVD	PMCRV	PMCRVD	PDSCRV	PDSCRVD	PExpCRV	PExpCRVD	PExcCRV	PExcCRVD	PTCRV	PTCRVD	PConCRV	PConCRVD	PComCRV	PComCRV	PSCV	PSCVD
Synapse		√	√	√	√	√				√	√	√	√	√									√	√
Velocity		√		√	√	√				√	√	√	√	√									√	√
Poi		√		√		√				√		√		√				√				√	√	√
Xalan				√	√	√		√		√	√	√	√	√									√	√
Camel				√	√							√	√										√	√
Ant		√	√	√	√	√				√	√	√	√	√									√	√

Table 7.28: Selected attributes from CSV and CK suites with fault-proneness for all systems

Both Coding Standard Violations-Based and CK Metrics																														
	Coding standard violations-based metrics																									CK metrics					
	PSRV	PSRVD	PNCRV	PNCRVD	PPCICRV	PPCICRVD	PFCRV	PFCRVD	PMCRV	PMCRVD	PDSCRV	PDSCRVD	PExpCRV	PExpCRVD	PExcCRV	PExcCRVD	PTCRV	PTCRVD	PConCRV	PConCRVD	PComCRV	PComCRVD	PSCV	PSCVD	WMC	DIT	NOC	CBO	RFC	LCOM
Synapse		√	√	√	√	√				√	√	√	√	√									√	√				√	√	√
Velocity		√	√	√	√	√				√	√	√	√	√									√	√				√	√	√
Poi		√		√	√	√				√	√	√	√	√									√	√				√	√	
Xalan		√		√	√	√		√		√	√	√	√	√									√		√			√	√	√
Camel		√		√	√	√				√	√	√	√	√									√	√				√	√	√
Ant		√	√	√	√	√				√	√	√	√	√									√	√				√	√	√

7.3.2 Multivariate Analysis With Respect To Fault Proneness

As explained in Section 7.3, the results characteristics will be discussed based on the goodness of fit in addition to the prediction accuracy. To evaluate the goodness of fit, we used the -2log likelihood, Cox & Snell R^2 and Nagelkerke R^2 measures. Table 7.29 presents those measures for each model in all systems under study.

Table 7.29: Multivariate Logistic Regression Analysis for all systems (Goodness of fit)

	-2log likelihood			Cox& Snell R^2			Nagelkerke R^2		
	Both	CSV	CK	Both	CSV	CK	Both	CSV	CK
Synapse	**253.883**	273.302	265.924	**0.248**	0.189	0.212	**0.344**	0.262	0.294
Velocity	**240.942**	253.23	266.308	**0.206**	0.162	0.113	**0.285**	0.225	0.156
Poi	419.535	**417.318**	495.886	0.294	**0.300**	0.160	0.404	**0.411**	0.220
Xalan	**1038.456**	1047.93	1149.117	**0.178**	0.169	0.067	**0.237**	0.225	0.089
Camel	**885.648**	891.68	928.272	**0.051**	0.045	0.007	**0.081**	0.071	0.011
Ant	**577.591**	588.891	598.945	**0.248**	0.236	0.226	**0.378**	0.360	0.344
Avg.	**569.343**	578.726	617.409	**0.204**	0.184	0.131	**0.288**	0.259	0.186

Keys: **Bolded** numbers indicate the best obtained value among the three.
Both means the model is built over a subset of both the coding standard violations-based and CK metrics
CSV means the model is built over a subset of the coding standard violations-based metrics only.
CK means the model is built over a subset of the CK metrics only.

7.3.2.1 Goodness of Fit

The goodness of fit section evaluate how well each model performs when applied to the data set it was derived from. For measuring the goodness of fit of linear regression model, the following two measures are used.

2log likelihood (-2LL**):** The -2LL for a model indicates the extent to which the model fails to perfectly predict the values of the dependent variable. -2LL is analogous to the Error Sums of Squares, SSE, in OLS regression. Results in Table 7.29 for -2LL show that except for Poi systems, the models built over subset of both have better goodness of fit than the ones built over subset of only CK metrics or coding standard violations-based

metrics. In poi system, the model built over only a subset of coding standard violations-based metrics has better goodness of fit than the other models.

Cox& Snell R^2: Cox & Snell R^2 is referred to as a "pseudo-R" statistic which designed to tell us something similar to what R^2 tells us in ordinary least-squares regression, that is the proportion of variance accounted for in the dependent variable based on the predictive power of the independent variables (predictors) in the model. The higher values of this measure are always better than the lower values so, results in Table 7.29 for Cox& Snell R^2 show that except for Poi systems, the models built over subset of both CSV and CK metrics have better goodness of fit than the ones built over subset of only CK metrics or coding standard violations-based metrics. In poi system, the model built over only a subset of coding standard violations-based metrics has better goodness of fit than the other models.

Nagelkerke R^2: Nagelkerke R^2 is another descriptive measure of goodness of fit. This measure is a variation of the R^2 concept defined for the OLS regression model. Like in linear regression, Nagelkerke R^2 tells us what percentage of the variability in the dependent variable can be explained by the predictors in the model. The higher values of this measure are always better than the lower values so, results in Table 7.29 for Nagelkerke R^2 show also that except for Poi systems, the models built over subset of combination of both CSV and CK metrics have better goodness of fit than the ones built over subset of only CK metrics or coding standard violations-based metrics. In poi system, the model built over only a subset of coding standard violations-based metrics has better goodness of fit than the other models. This observation is expected due to the

amount of variation in dependent variable, explained by the inclusion of more independent variable (see Table 7.28).

In general, the results presented in Table 7.29 show that the models built upon subset of combination of both coding standard violations-based metrics and CK metrics have always a better goodness of fit than those models built upon on a subset of either coding standard violations-based metrics or CK metrics. The exception only in the Poi systems in which the model built up on only a subset of coding standard violations-based metrics has a better goodness of fit than the other models. Comparing CK with coding standards violation-based metrics, the results show that except for Synapse, the models built upon coding standard violation-based metrics have a better goodness of fit than the models built upon CK metrics in all systems investigated by this study.

To sum up, it can be observed that the model fit measures indicate that the inclusion of the coding standard violations-based metrics significantly enhanced the goodness of fit of the logistic regression models for predicting the fault-proneness of classes.

7.3.2.2 Model's Prediction Accuracy

Regarding the model's prediction accuracy, the predictive accuracy of the prediction models is evaluated using the correct classification rate which is the ratio of the number of classes that were correctly classified (predicted) as fault-prone or non-fault-prone to the total number of classes. Additionally, another accuracy measure called the receiver operating characteristic (ROC) curve is used. ROC curve area is a graphical plot which illustrates the performance of a binary classifier as its discrimination threshold

is varied. The higher values of both correct classification rate and ROC curve area are always better.

The results of correct classification rate and ROC curve area for the three models are presented in Table 7.30 and 7.31 respectively. Regarding the correct classification rate, if the average classification rate is considered, the model built upon subsets of both coding standard violations-based metrics and CK metrics achieved the highest correct classification rate among the three models.

On the other hand, if the number of times where each model achieves the best correct classification rate is counted, the following can be observed. In synapse system, the model built upon subsets of both suites of metrics and the model built upon a subset of coding standard violations-based metrics achieved the same correct classification rate. The model built upon a subset of CK metrics achieved the highest correct classification rate in only one system which is Velocity system. Moreover, the model built upon subsets of both suites of metrics achieved the highest correct classification rate in Synapse and Poi systems, while the model built upon a subset of coding standard violations-based metrics achieved the highest correct classification rate in Synapse, Xalan, Camel and Ant.

Table 7.30: Correct classification rate for the three models in all systems

System	Correct classification rate			#of instances
	Both	CSV	CK	
Synapse	**71.484**	**71.484**	69.922	256
Velocity	63.756	64.629	**68.122**	229
Poi	**69.932**	55.353	39.408	439
Xalan	66.057	**66.400**	59.543	875
Camel	77.706	**77.921**	72.883	933
Ant	79.352	**82.456**	80.297	741
Avg.	**71.381**	69.707	65.029	

Keys: **Bolded** numbers indicate the best obtained value among the three.
Both means the model is built over a subset of both the coding standard violations-based and CK metrics
CSV means the model is built over a subset of the coding standard violations-based metrics only.
CK means the model is built over a subset of the CK metrics only.

Table 7.31: ROC curve area for the three models in all systems

System	ROC Curve Area			#of instances
	Both	CSV	CK	
Synapse	0.735	0.732	**0.743**	256
Velocity	0.704	0.694	**0.71**	229
Poi	0.799	**0.804**	0.794	439
Xalan	0.691	**0.702**	0.613	875
Camel	0.595	**0.599**	0.543	933
Ant	0.810	**0.836**	0.823	741
Avg.	0.722333	**0.727833**	0.704333	

Keys: **Bolded** numbers indicate the best obtained value among the three.
Both means the model is built over a subset of both the coding standard violations-based and CK metrics
CSV means the model is built over a subset of the coding standard violations-based metrics only.
CK means the model is built over a subset of the CK metrics only.

In addition to the correct classification rate, the predictive accuracy of the prediction models is also evaluated using ROC curve area. ROC curve area is a graphical plot which illustrates the performance of a binary classifier as its discrimination threshold is varied. The higher values of ROC are always better, so the highest achieved ROC values among the three models are marked in boldface values in Table 7.31.

As shown in Table 7.31, if the average ROC curve area is considered, the model built upon a subset of coding standard violations-based metrics achieved the highest ROC curve area among the three models. On the other hand, if we count the number of times where each model achieves the best ROC curve area, we get the following. The model built upon a subset of coding standard violations-based metrics achieved the highest ROC curve area four times in Poi, Xalan, Camel and Ant systems. The model built upon a subset of CK metrics achieved the highest ROC curve area two times in Synapse and Velocity systems. Additionally, if the categorization of ROC curve area adopted by [46] is considered, the model built upon subsets of both coding standard violations-based metrics and CK metrics falls into "Acceptable" category three times in Synapse, Velocity and Poi systems. Also the same model falls into "Excellent" category one time in Ant system and falls into "Poor" one time in Camel system and fall into "Fair" one time in Xalan system. Regarding the model built upon a subset of only coding standard violations-based metrics, it falls into "Excellent" category two times in Poi and Ant systems. Also the same model falls into "Acceptable" category two times in Synapse and Xalan systems and one time into "Fair" in Velocity system and one time into "Poor" in Camel system. Finally, the model built upon a subset of only CK metrics falls into "Excellent" category one time in Ant system. Also the same model falls into "Acceptable" category three times in Synapse, Velocity and Poi systems and one time into "Fair" and "Poor" categories in Xalan and Camel respectively.

7.3.2.2.1 Validating Hypothesis H2

Tow prediction accuracy measures are used to validate the hypothesis H2. Those accuracy measures are discussed as follows:

- **Correct classification rate:** Comparing the performance of the regression models based on the coding standard violations-based metrics against those based on the CK metrics in predicting the fault-proneness of classes, the results show the following. In term of the number of times, where each model achieved the best correct classification rate, the coding standard violations-based metrics achieved the best correct classification rate five times in Synapse, Poi, Xalan, Camel and Ant systems while the CK metrics only achieved the best correct classification rate one time in Velocity system. In terms of the average, the coding standard violations-based metrics provide better result than CK metrics. So, this provides an evidence to accept hypothesis H2.
- **ROC curve area:** Comparing the performance of the regression models based on the coding standard violations-based metrics against those based on the CK metrics in predicting the fault-proneness of classes, the results show the following. In term of the number of times, where each model achieves the highest ROC curve area, the coding standard violations-based metrics achieved the highest ROC curve area four times in Poi, Xalan, Camel and Ant systems while the CK metrics only achieved the highest ROC curve area two times in Synapse and Velocity systems. In terms of the average, the coding standard violations-based metrics provide better result than CK metrics. This provides an evidence to accept hypothesis H2.

7.3.2.2.2 Validating Hypothesis H3

Tow prediction accuracy measures are used to validate the hypothesis H3. Those accuracy measures are discussed as follows:

- **Correct classification rate**: Comparing the performance of the regression models based on both suites against those based on the CK metrics only, in predicting the fault proneness of classes, the results show the following. In terms of the number of times, where the model achieves the best correct classification rate, the regression models built upon subsets of both suites achieved the best correct classification rate four times in Synapse, Poi, Xalan and Camel systems, while the regression model built upon only the CK metrics achieved the best correct classification rate two times in Velocity and Ant systems. In terms of the average, the regression models built upon subsets of both suites also provide better results than the CK metrics. This provides an evidence to accept hypothesis H3.
- **Roc curve area**: Comparing the performance of the regression models based on both suites against those based on the CK metrics only, in predicting the fault proneness of classes, the results show the following. In terms of the number of times, where the model achieves the highest ROC curve area, the regression models built upon subsets of both suites achieved the highest ROC curve area three times in Poi, Xalan and Camel systems, while the regression model built upon only the CK metrics achieved the highest ROC curve area three times in Synapse, Velocity and Ant systems. In terms of the average, the regression models built upon subsets of both suites provide better results than the CK metrics. So, this provides an evidence to accept hypothesis H3.

7.3.2.2.2 Validating Hypothesis H4

Tow prediction accuracy measures are used to validate the hypothesis H4. Those accuracy measures are discussed as follows:

- **Correct classification rate**: Comparing the performance of the regression models based on both suites against those based on the coding standard violations-based metrics only, in predicting the fault proneness of classes, the results show the following. In terms of the number of times, where the model achieves the best correct classification rate, both types of models achieved equal results in synapse system. The regression models built upon subsets of both suites of metrics achieved the best correct classification rate only one time in Poi system, while the regression model built upon only the coding standard violations-based metrics achieved the best correct classification rate four times in Velocity, Xalan, Camel and Ant systems. In terms of the average, the regression models built upon subsets of both suites of metrics provide better results than the coding standard violations-based metrics. This provides no clear evidence to either accept or reject the hypothesis H4.
- **ROC**: Comparing the performance of the regression models based on both suites against those based on the coding standard violations-based metrics only, in predicting the fault proneness of classes, the results show the following. In terms of the number of times, where the model achieves the highest ROC curve area, the regression models built upon subsets of both suites achieved the highest ROC curve area only two times in Synapse and Velocity systems, while the regression models built upon only the coding standard violations-based metrics achieved

highest ROC curve area four times in Poi, Xalan, Camel and Ant systems. In terms of the average, the regression models built upon a subset of the coding standard violations-based metrics also provide better results than the models built upon subsets of both suites. This provides an evidence to reject hypothesis H4.

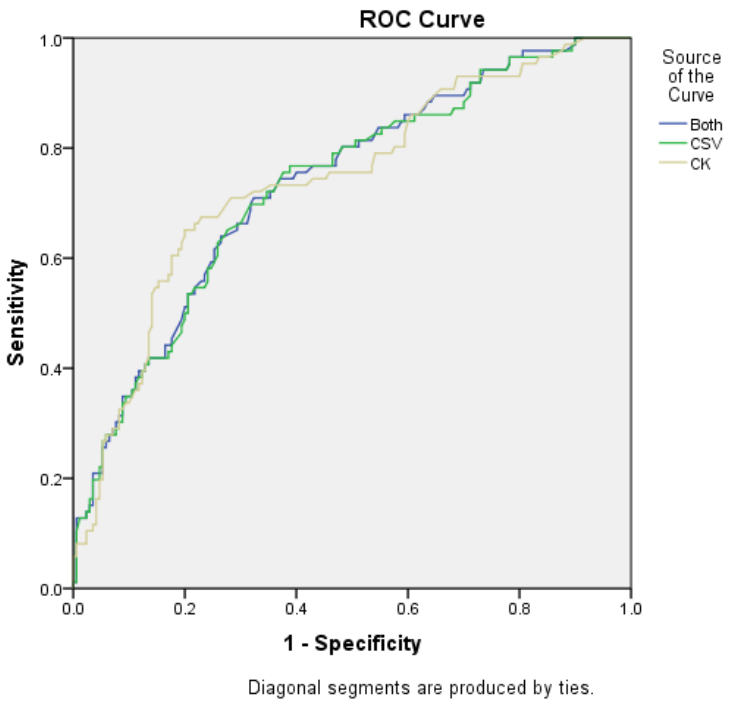


Figure 7.1: Roc curve area using Synapse data

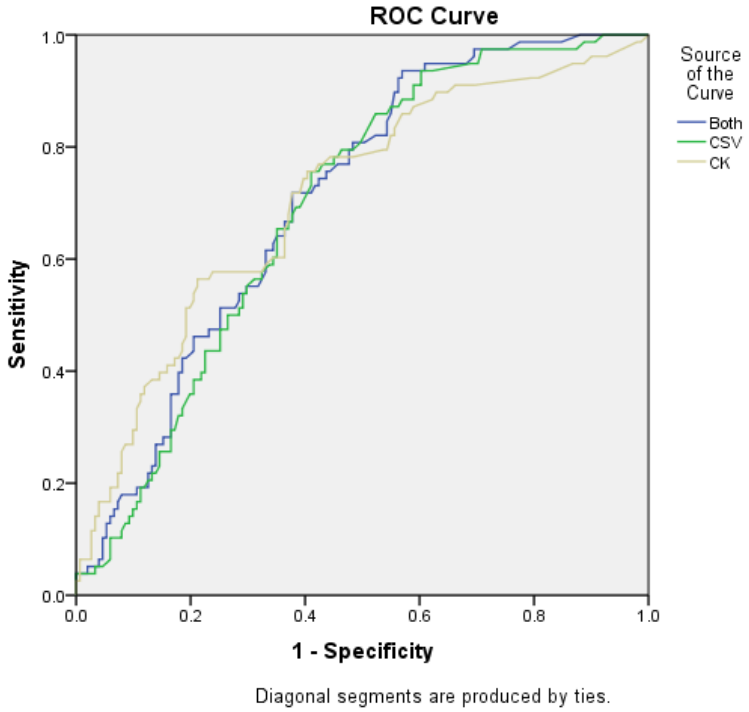


Figure 7.2: Roc curve area using Velocity data

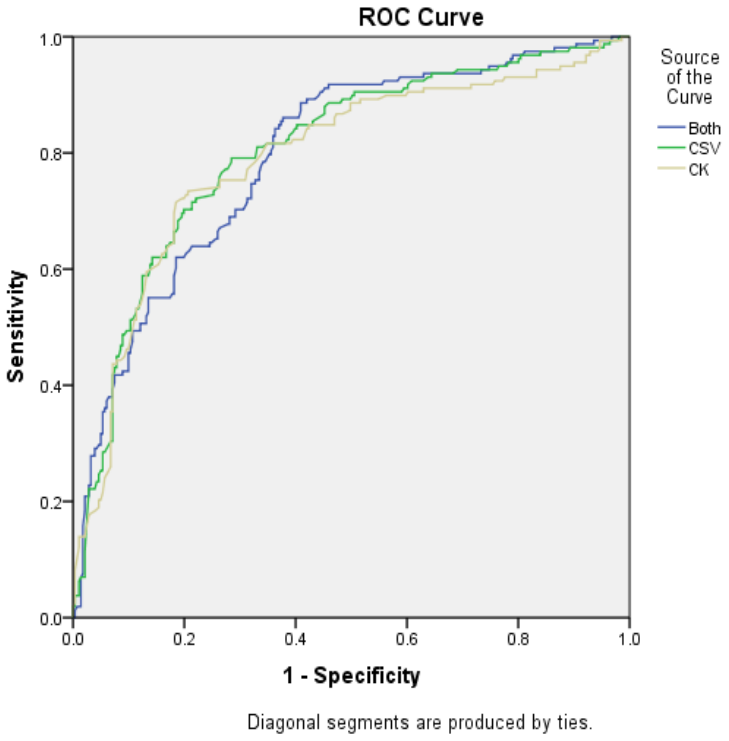


Figure 7.3: ROC curve area using Poi data

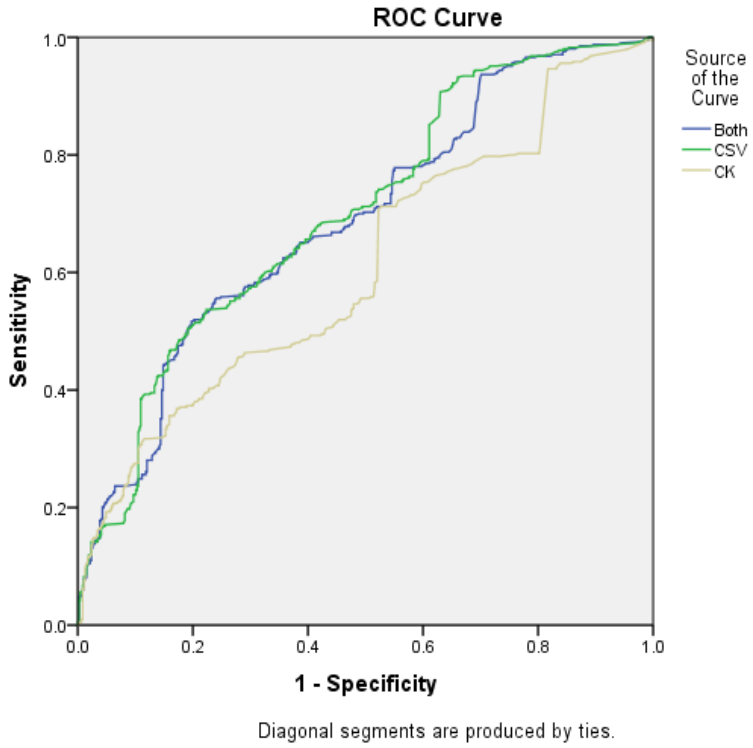


Figure 7.4: ROC curve area using Xalan data

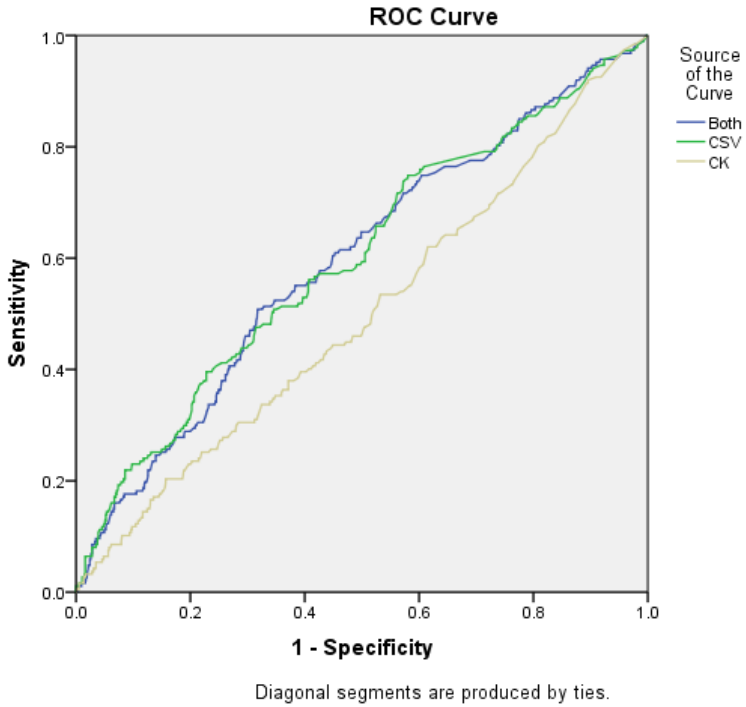


Figure 7.5: ROC curve area using Camel data

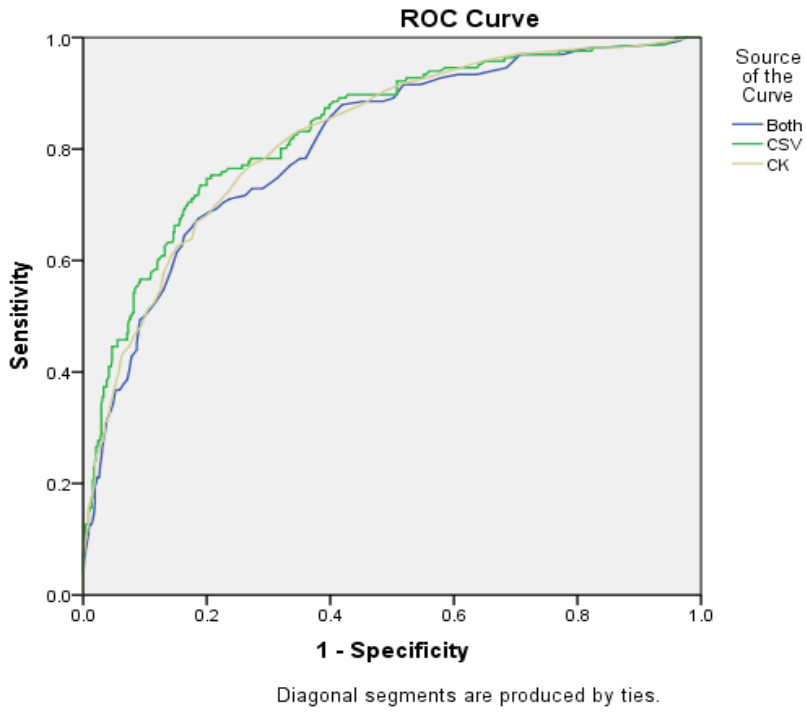


Figure 7.6: ROC curve area using Ant data

7.3.3 Attribute Selection With Respect To Fault Density

Results of correlation and PCA show that some of the metrics capture similar dimensions in the data set which reflecting the fact that some of them contain redundant or noisy information. Therefore, to build accurate models to predict the fault density in the class, attribute selection procedure is performed as described in Section 6.3.4. The results of the attribute selection against the fault density, as dependent variable for all systems under study, are presented in Tables 7.32, 7.33 and 7.34.

As shown in these tables, some metrics are selected more than one time as fault density predictors across systems under study, some of them are selected only once, and some others have never been selected in all systems under study. The coding standard violations-based metrics that have never been selected, at any system from the target set of this study, as predictors for the fault-proneness of the class, are PSRV, PNCRV, PPCICRV, PFCRV, PDSCRV, PExpCRV, PExpCRVD, PExcCRV PTCRV and PSCV. The reason could be the high correlation of these metrics with some other metrics in all systems under study (Tables E.1 through E.6 in the appendix E). Another reason could be that, in the systems investigated in this research, these metrics might not be useful indicators of the fault density of the class. Thus, they are removed from the linear regression models. However, it cannot be generalized that these metrics are not useful predictors for the fault density of the class in other case studies.

Regarding the CK metrics, the metrics that have never been selected as predictors for the fault density of the class in all systems under study is Lcom. So this metric is removed from the linear regression models.

Table 7.32: Selected attributes from CK metrics with fault-density

CK Metrics						
	WMC	DIT	NOC	CBO	RFC	LCOM
Synapse		√				
Velocity		√				
Poi		√	√		√	
Xalan		√		√		
Camel		√			√	
Ant	√	√	√			

Table 7.33: Selected attributes from coding standard violations-based suite with fault density for all systems

Coding Standard Violations-Based Metrics																								
	PSRV	PSRVD	PNCRV	PNCRVD	PPCICRV	PPCICRVD	PFCRV	PFCRVD	PMCRV	PMCRVD	PDSCRV	PDSCRVD	PExpCRV	PExpCRVD	PExcCRV	PExcCRVD	PTCRV	PTCRVD	PConCRV	PConCRVD	PComCRV	PComCRV	PSCV	PSCVD
Synapse										√														√
Velocity		√				√		√																√
Poi																								√
Xalan										√						√				√				√
Camel		√		√		√			√			√						√	√			√		√
Ant								√																√

Table 7.34: Selected attributes from CSV and CK suites with fault density for all systems

Both Coding Standard Violations-Based and CK Metrics																														
	Coding standard violations-based metrics																									CK metrics					
	PSRV	PSRVD	PNCRV	PNCRVD	PPCICRV	PPCICRVD	PFCRV	PFCRVD	PMCRV	PMCRVD	PDSCRV	PDSCRVD	PExpCRV	PExpCRVD	PExcCRV	PExcCRVD	PTCRV	PTCRVD	PConCRV	PConCRVD	PComCRV	PComCRVD	PSCV	PSCVD	WMC	DIT	NOC	CBO	RFC	LCOM
Synapse										√														√		√				
Velocity		√				√		√																√		√				
Poi						√																		√		√				
Xalan										√						√				√				√						
Camel		√		√		√			√			√						√	√		√			√		√				
Ant								√																√			√			

7.3.4 Multivariate Analysis With Respect To Fault Density

The results characteristics will be discussed based on the goodness of fit in addition to the prediction accuracy. To evaluate the goodness of fit, R^2 and adjusted R^2 measures. Table 7.34 presents those measures for each model in all systems under study.

Table 7.34: Multivariate Linear Regression Analysis for all systems (Goodness of fit)

	R^2			Adjusted R^2		
	Both	CSV	CK	Both	CSV	CK
Synapse	**0.054**	0.053	0.000	0.043	**0.046**	-0.004
Velocity	**0.023**	0.014	0.008	0.001	-0.004	**0.004**
Poi	**0.064**	0.039	0.013	**0.058**	0.036	0.006
Xalan	**0.013**	**0.013**	0.007	**0.009**	**0.009**	0.005
Camel	**0.031**	**0.031**	0.001	0.022	**0.023**	-0.001
Ant	**0.004**	0.003	**0.004**	0.000	0.000	0.000
Avg.	**0.032**	0.026	0.006	**0.023**	0.0197	0.003

Keys: **Bolded** numbers indicate the best obtained value among the three.
Both means the model is built over subsets of both the coding standard violations-based and CK metrics
CSV means the model is built over a subset of the coding standard violations-based metrics only.
CK means the model is built over a subset of the CK metrics only.

7.3.4.1 Goodness of Fit

The goodness of fit section evaluate how well each model performs when applied to the data set it was derived from. For measuring the goodness of fit of linear regression model, we used the following two measures.

R^2 and adjusted R^2: these two measures of goodness of fit tell us what percentage of the variability in the dependent variable can be explained by the predictors in the model. The higher values of these two measures always indicate a better fit than the lower values. Also it is known that adjusted R^2 is often smaller than R^2. The results for these two measures for the built linear regression models are presented in Table 7.34 for all systems under study. The goodness of fit results show that almost all R^2 and adjusted R^2 values

are relatively small in the target set of systems which implies that the models have low goodness of fits. The results presented in Tables 7.34 confirm what stated by [54]: “for technical reasons, high values of R^2 are rare, even for accurate models”. Furthermore, such observation might be due to the concentration of faults distribution presented in Table 6.1.

In general, the results presented in Table 7.34 show the following. Regarding R^2, if we count the number of times where the model fits better than the other models, the models built upon subsets of both coding standard violations-based metrics and CK metrics are equivalent to the models built upon only a subset of coding standard violations-based metrics two times in Xalan and Camel systems and equivalent to the models built upon only a subset of CK metrics one time in Ant system. But in the other three systems, the models built upon both suites fit better than the models built Coding standards metrics and also the models built up on the CK metrics. In terms of the average, the linear regression models built upon subsets of both suites also record the highest average value of R2 which indicate that they are fitting better than the other models.

Comparing CK with coding standards violation-based metrics, the results in Table 7.34 show that except for Ant system, the models built upon coding standard violation-based metrics have a better goodness of fit than the models built upon CK metrics in all systems investigated by this study.

To sum up, the model fit measures indicate that the inclusion of the coding standard violations-based metrics significantly enhanced the goodness of fit of the linear regression models for predicting the fault density of classes.

7.3.4.2 Model's Prediction Accuracy

To obtain a realistic evaluation of the predictive power of the coding standard violations-based metrics compared to the CK metrics, cross validation technique was performed in three different ways:

- Using subset of both coding standard violations-based metrics and CK metrics as independent variables.
- Using only subset of coding standard violations-based metrics as independent variables.
- Using only subset of CK metrics as independent variables.

Each subset for each system under study was selected according to the attribute selection procedure. The results' characteristics are addressed based on the mean absolute error (MAE) which was used as accuracy measure. The results are summarized in Tables 7.35 for all systems under study. To compare the accuracy of the three different types of the models, the average and the number of times, where each type of the models achieved the smallest error value, were considered as comparison factors among the models for the accuracy measure.

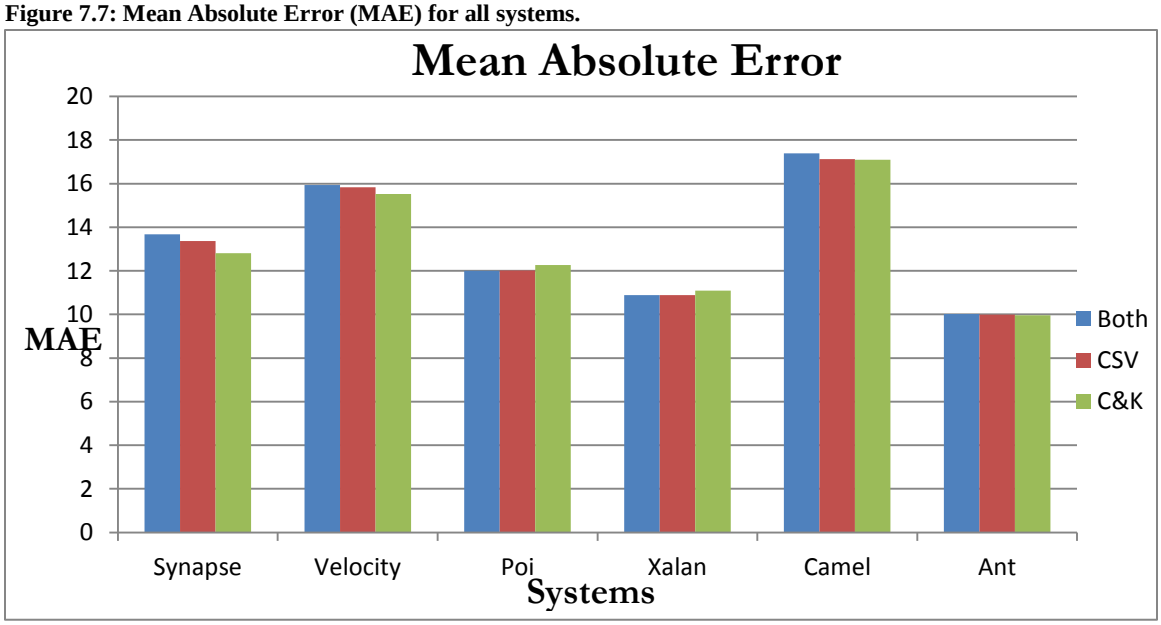
Table 7.35: Prediction accuracy for linear regression models (all systems) using MAE

System	Mean Absolute Error (MAE)			#of instances
	Both	CSV	CK	
Synapse	13.6676	13.3683	**12.8068**	256
Velocity	15.9371	15.8304	**15.5209**	229
Poi	11.9991	**12.0087**	12.2587	439
Xalan	10.889	**10.8884**	11.0975	875
Camel	17.3922	17.1257	**17.1009**	933
Ant	10.0219	9.9876	**9.9656**	741
Avg.	13.31782	13.20152	**13.12507**	

Keys: **Bolded** numbers indicate the best obtained value among the three.
Both means the model is built over subsets of both the coding standard violations-based and CK metrics
CSV means the model is built over a subset of the coding standard violations-based metrics only.
CK means the model is built over a subset of the CK metrics only.

As shown in Table 7.35, except for Poi and Xalan systems, the linear regression models built up on a subset of CK metrics have better MAEs values than the regression models built upon either subsets of both suites or a subset of coding standard violations-based metrics in all systems under study. Also the regression models built up on a subset of coding standard violations-based metrics achieved the best values of MAE two times in Poi and Xalan systems. Furthermore, the regression models built upon subsets of both suites achieved the worst MAE values in all systems. In terms of the average, the models built upon a subset of CK metrics have better MAE values than those built upon either coding standard violations-based metrics or both suites.

Figure 7.7: Mean Absolute Error (MAE) for all systems.



7.3.4.2.1 Validating Hypothesis H6

To validate the hypothesis H6, this research work used a prediction accuracy measure called mean absolute error MAE. This accuracy measure is discussed as follows:

MAE: Comparing the performance of the regression models built upon CK metrics against the regression models built upon the coding standard violations-based metrics in predicting the fault density of classes, the results show the following. In term of the number of times, where each model achieves the smallest error, the models built upon a subset of the CK metrics achieved the smallest error four times in Synapse, Velocity, Camel and Ant systems, while the models built upon a subset of the coding standard violations-based metrics achieved the smallest error only two times in Poi and Xalan systems. In terms of the average, also the models built upon a subset of the CK metrics achieved the smallest average MAE. Thus, there is an evidence to reject hypothesis H6.

7.3.4.2.2 Validating Hypothesis H7

To validate the hypothesis H7, this research work used a prediction accuracy measure called mean absolute error MAE. This accuracy measure is discussed as follows:

MAE: Comparing the performance of the regression models built upon CK metrics against the regression models built upon both suites in predicting the fault density of classes, the results show the following. In term of the number of times, where each model achieves the smallest error, the models built upon a subset of the CK metrics achieved the smallest error four times in Synapse, Velocity, Camel and Ant systems, while the models built upon both suites of metrics achieved the smallest error only two times in Poi and Xalan systems. In terms of the average, also the models built upon a subset of the CK metrics achieved the smallest average MAE. Thus, there is an evidence to reject hypothesis H7.

7.3.4.2.2 Validating Hypothesis H8

To validate the hypothesis H8, this research work used a prediction accuracy measure called mean absolute error MAE. This accuracy measure is discussed as follows:

MAE: Comparing the performance of the regression models built upon a subset of coding standard violations-based metrics against the regression models built upon both suites of metrics in predicting the fault density of classes, the results show the following. In term of the number of times, where each model achieves the smallest error, the models built upon a subset of the coding standard violations-based metrics achieved the smallest error in all systems under study, while the models built upon both suites of metrics failed to do so in all systems. In terms of the average, also the models built upon a subset of the

coding standard violations-based metrics achieved the smallest average MAE. Thus, there is an evidence to reject hypothesis H8.

Table 7.36 Hypotheses results summary

Hypothesis	Accepted	Rejected	Partially accepted
H1			√
H2	√		
H3	√		
H4		√	
H5			√
H6		√	
H7		√	
H8		√	

7.4 Threats to Validity

The findings in this research have a number of limitations that are not unique to our study but are common with most of the empirical studies in the literature.

7.4.1 Construct Validity

"Construct validity is the degree to which the independent and dependent variables actually quantify the concepts they supposed to measure" [53]. Fault proneness and fault density are used as inverse proxies for the functional correctness quality attribute addressed by this research. The faults types and faults severity are not considered because such information is not available in repository used to acquire the faults data. One of the dependent variables is the fault proneness is a binary variable which set to 0 if the class has no faults or 1 otherwise. Additionally, this research work focused on the top-level classes. Inner classes were considered as contents included in the outer class. The proposed coding standards violations based metrics (independent variables) were empirically validated and their practicality and usefulness are compared

with CK metrics. Despite the availability of coding standards violations metrics proposed in the literature, this research work compared the proposed metrics with CK metrics because they are supported by many tools, they are well defined in the literature and they have been empirically and theoretically validated with different quality attributes.

7.4.2 Internal Validity

The collected coding rules violations depend on the usage of static analyzers or code formatters during the development of the systems. The chance of coding rules violations to be occurred are expected to increase with absence of such code analyzers and formatters. In this study, the findings are based on the correlation and regression analysis performed on the dataset collected from the target set of systems. Additionally, the introduction of coding rules violations may increase or decrease according to the experience of the developers of the systems under study. Association between most of the proposed metrics and both the fault-proneness and the fault density was confirmed, but causality of the association cannot be claimed. Furthermore, three java dedicated static analysis tools are used for inspecting java source code to find the violations of coding standard rules. Although these tools are commonly used and recommended by the developers of JPL standard, there is no official and reliable technical report about the false negatives and positives rates of these tools. So the generated violations reports are taken as they are without considering such false positives and negatives. Finally, in this research only the distinct rules violations aspect is considered. However, other aspects of such violations like violations density and severity are not taken into account. Furthermore, in this research the semantics of some rules such as the naming rule which emphasis to make meaningful names of identifiers. Such threat is ignored due to the lack

of such feature support in the used tools. Additionally, 85% of the JPL coding standard's rules are comprehend due to the lack of tools' support for the remained 15% of the selected coding standard.

7.4.3 External validity

The first external threat to validity is that all systems understudy are written in Java programming language. Other object oriented programming languages may have different features than those in java. Second, all systems considered by this study are open source software, which may not be good representative of all industrial domain applications. Third, although the six systems, from which the data are collected, are reasonable in size in terms of the number of classes, and also they belong to different application domain, the obtained results cannot be generalized. Further investigation is required with different systems to confirm the findings and draw stronger conclusions.

7.4.4 Conclusion validity

In this research work, non-parametric tests are used instead of the parametric tests. Although the power of parametric tests is generally higher than the power of non-parametric tests, this research used non-parametric tests because the distribution of parameters cannot be assumed. Additionally, in multivariate regression analysis results, the number of times where the built model achieved the best results is counted. The dataset involves six systems and the evaluation is done according to these six systems. But, it may not the case if additional systems are used. So, more systems should be used to report the tendency of the results.

CHAPTER 8

CONCLUSION AND FUTURE WORK

This thesis investigates the relationships between the coding standards violations at the class level and the software functional correctness quality attribute in terms of fault proneness and fault density. A set of coding standard violations-based metrics have been derived according to the coding rules categorization proposed and adopted by JPL coding standard for java programming language. The proposed metrics are empirically validated separately using (univariate modeling) and in combination using (multivariate modeling). This empirical analysis involves classes from six java open source projects.

The univariate regression analysis is applied to explore the abilities of the proposed metrics individually to predict both fault proneness and fault density aspects of the functional correctness quality attribute. The multivariate regression analysis is applied to investigate the abilities of the coding standard violations-based metrics combination to predict fault proneness and fault density and to build the corresponding practical models.

The multivariate regression models were built in three different ways:

- Using subset of both coding standard violations-based metrics and CK metrics as independent variables.
- Using only subset of coding standard violations-based metrics as independent variables.
- Using only subset of CK metrics as independent variables.

The main findings derived from the statistical analysis performed in this research can be summarized as follows. The correlation analysis showed that many of the proposed coding standard violations-based metrics have a relationship with faults and can be used as potential indicators to identify fault-prone classes. Additionally, the correlation analysis also showed that some of the proposed metrics can be considered as useful predictors to estimate the fault density of object oriented classes. The principle component analysis showed that many of the proposed coding standard violations-based metrics fall into the first two components which in turn reflects the importance of these metrics. Also the principle component analysis showed consistent result with correlation analysis results. Ranging from "fair" to "excellent", the univariate logistic regression analysis showed that some of the proposed metrics represent significant and practical predictors for fault prone classes, while the others represent insignificant or poor predictors. The constructed multivariate models were better than most univariate models in their abilities to predict class fault proneness and fault density. The multivariate regression analysis showed that the inclusion of the proposed coding standard violations-based metrics to the prediction models improved the prediction accuracy for class fault-proneness compared to the models that only consider CK metrics. The coding standard violations-based metrics showed competitive results compared to CK metrics for both class fault-proneness and the fault density prediction in all systems under study.

The results showed that not only the product structural properties measured by CK metrics are important quality indicators, but also the coding standards have the same level of importance. So, they should be adopted and considered during the software development process. Among the coding rules studied by this research, naming,

expressions, fields, complexity, packages, classes, interfaces, declarations and statements were found to be correlated with software faults. Those are examples of such coding rules: use the standards naming conventions, do not override field or class names, make imports explicit, do not have cyclic package and class dependencies, obey the contract for equals() method, define both equals() and hashCode() methods, do not implement the Cloneable interface, do not call nonfinal methods in constructors, make fields private, do not use static mutable fields, use braces in control structures, do not have empty blocks, use breaks in switch statements, use named constants for non-trivial literals, make operator precedence explicit and so on. So, the software projects' managers should ensure that those rules are adhered to the extreme and should also find automated tools to enforce them. Additionally, such tools should be made as part of the developers' integrated development environment. Furthermore, software projects' managers should restrict the acceptance to those code modules that has passed the automated checking for the above mentioned rules.

8.1 Research Contributions

The contribution of this research is as follows.

- Surveying the literature for identifying the existing coding standards violations and adherence metrics for different software quality attributes prediction.
- Set of coding standard violations-based metrics was derived and validated empirically as potential predictors for the fault proneness and fault density of the classes of object oriented systems.

- Three types of multivariate logistic regression models (i.e. based on subset of CK metrics, subset coding standard violations-based metrics, and subset of both) were built for classifying the classes into fault-prone or non-fault-prone.
- Three types of multivariate linear regression models (i.e. based on subset of CK metrics, subset coding standard violations-based metrics, and subset of both) were built for predicting the fault density of classes.
- The performance of product and coding standard violations-based metrics were compared as predictors for fault-proneness of the classes of object-oriented systems.
- The performance of product and coding standard violations-based metrics were compared as predictors for the fault density of the classes of object-oriented systems.

8.2 Future Work

Directions for future work related to the contribution of this research work can be outlined as follows.

- **Validating the metrics using additional open source and commercial software projects:** the characteristics of the system under study are believed to be an important factor to evaluate which one of the two suites is better in identifying fault-prone classes and in estimating fault density of objects oriented classes.
- **Validating the practicality and usefulness of the proposed metrics in predicting other software quality attributes:** this research work investigated the practicality and usefulness of coding standards violations-based metrics as

indicators or predicators for functional correctness quality attribute in terms of fault-proneness and fault density. Other software quality attributes can be investigated by further research using the coding standards violations-based metrics as indicators for such quality attributes.

- **Studying the potential impacts of additional publicized coding standards and self-imposed (commercial) coding standards on different software quality attributes:** this research work studied the potential impacts of a well-publicized coding standard called JPL coding standard for java programming language on functional correctness in terms of fault-proneness and fault density. Further studies can be performed to address the relationships between other standards (public and commercial) and software quality attributes.
- **Studying the practicality and usefulness of the metrics using non-parametric techniques:** this research work investigated the usefulness of the coding standards violations-based metrics using the statistical parametric techniques like regression. Another investigation can be performed using non-parametric techniques like neural networks.

References

[1] Boogerd, C. and L. Moonen. *Assessing the value of coding standards: An empirical study*. in *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*. 2008.

[2] Mohan, A. and N. Gold. *Programming style changes in evolving source code*. in *Program Comprehension, 2004. Proceedings. 12th IEEE International Workshop on*. 2004.

[3] Oman, P.W. and C.R. Cook, *Typographic style is more than cosmetic.* Commun. ACM, 1990. 33(5): p. 506-520.

[4] Henricson, M. and E. Nyquist. *Programming in C++, rules and recommendations*. 1992 [cited 2013; Available from: citeseer.ist.psu.edu/henricson92programming.html.

[5] Miryung, K., et al. *An ethnographic study of copy and paste programming practices in OOPL*. in *Empirical Software Engineering, 2004. ISESE '04. Proceedings. 2004 International Symposium on*. 2004.

[6] Pfleeger, S.L., *Software Engineering: The Production of Quality Software*. 1991: Macmillan Publishing Company.

[7] Succi, G. and M. Marchesi, eds. *Extreme programming examined*. 2001, Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA,.

[8] Halloran, T.J. and W.L. Scherlis, *High Quality and Open Source Software Practices*, in *Meeting Challenges and Surviving Success: The 2nd Workshop on Open Source Software Engineering*. 2003.

[9] Reddy, A., *Java coding style guide*. 1999.

[10] Emanuelsson, P. and U. Nilsson, *A Comparative Study of Industrial Static Analysis Tools.* Electronic Notes in Theoretical Computer Science, 2008. 217(0): p. 5-21.

[11] Porter, A.A. and R.W. Selby, *Empirically guided software development using metric-based classification trees.* Software, IEEE, 1990. 7(2): p. 46-54.

[12] Güneş Koru, A. and H. Liu, *Identifying and characterizing change-prone classes in two large-scale open-source products.* Journal of Systems and Software, 2007. 80(1): p. 63-73.

[13] Control, E.B.f.S.S.a., *Java coding Standard*. 2005: PARIS CEDEX, France.

[14] Li, W. and S. Henry, *Object-oriented metrics that predict maintainability.* Journal of Systems and Software, 1993. 23(2): p. 111-122.

[15] Card, D.N. and C.L. Jones. *Status report: practical software measurement*. in *Quality Software, 2003. Proceedings. Third International Conference on*. 2003.

[16] Fenton, N.E. and S.L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*. 1998: PWS Publishing Co. 656.

[17] Chidamber, S.R. and C.F. Kemerer, *Towards a metrics suite for object oriented design.* SIGPLAN Not., 1991. 26(11): p. 197-211.

[18] Boogerd, C. and L. Moonen. *Evaluating the relation between coding standard violations and faultswithin and across software versions*. in *Mining Software*

Repositories, 2009. MSR '09. 6th IEEE International Working Conference on. 2009.

[19] *Guidelines for the Use of the C Language in Critical Systems*. 2004.

[20] Basalaj, W. and F.v.d. Beuken, *Correlation Between Coding Standards Compliance and Software Quality*. 2006.

[21] *High Integrity C++ Coding Standard Manual*. 2004 [cited 2013 May 2013]; Available from: www.codingstandard.com.

[22] Kawamoto, K. and O. Mizuno. *Predicting Fault-Prone Modules Using the Length of Identifiers*. in *Empirical Software Engineering in Practice (IWESEP), 2012 Fourth International Workshop on*. 2012.

[23] Takai, Y., T. Kobayashi, and K. Agusa. *Software Metrics Based on Coding Standards Violations*. in *Software Measurement, 2011 Joint Conference of the 21st Int'l Workshop on and 6th Int'l Conference on Software Process and Product Measurement (IWSM-MENSURA)*. 2011.

[24] Elish, M.O. and J. Offutt., *The adherence of open source java programmers to standard coding practises.*, in *6:th IASTED International Conference on Software Engineering and Applications*. 2002: USA.

[25] Smit, M., Gergel, B, Hoover, H.J and Stroulia, E.,, *Maintainability and Source Code Conventions: An Analysis of Open Source Projects*, in *27th IEEE International Conference on , 2011*. 2011.

[26] Smit, M., et al. *Code convention adherence in evolving software*. in *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*. 2011.

[27] Butler, S., et al., *Exploring the Influence of Identifier Names on Code Quality: An Empirical Study*, in *Proceedings of the 2010 14th European Conference on Software Maintenance and Reengineering*. 2010, IEEE Computer Society. p. 156-165.

[28] Havelund, K. and A. Niessner, *JPL Java Coding Standard*. 2010, California Institute of Technology.

[29] *Semmle*. 2013 [cited 2013 Aug 2013]; Available from: http://semmle.com.

[30] Li, X. and C. Prasad, *Effectively teaching coding standards in programming*, in *Proceedings of the 6th conference on Information technology education*. 2005, ACM: Newark, NJ, USA. p. 239-244.

[31] Barr, M. *enforcing coding standards automatically*. 2009 [cited 2013 May 2013].

[32] SourceForge. *FindBugs*. 2013 [cited 2013 May 2013]; Available from: http://findbugs.sourceforge.net/.

[33] Rutar, N., C.B. Almazan, and J.S. Foster. *A comparison of bug finding tools for Java*. in *Software Reliability Engineering, 2004. ISSRE 2004. 15th International Symposium on*. 2004.

[34] SourceForge. *PMD*. 2013 [cited 2013 May- 2013]; Available from: http://pmd.sourceforge.net/.

[35] SourceForge. *Checkstyle*. 2013 [cited 2013 May 2013]; Available from: http://checkstyle.sourceforge.net/.

[36] Silberschatz, A., H. Korth, and S. Sudarshan, *Database Systems Concepts*. 2006: McGraw-Hill, Inc. 1168.

[37] Elmasri, R. and S. Navathe, *Fundamentals of Database Systems*. 2010: Addison-Wesley Publishing Company. 1200.
[38] Boslaugh, S. and P.A. Walters, *Statistics In A Nutshell: A Desktop Quick Reference*. 2008: O'Reilly Media.
[39] SciTools. *understand for Java*. 2013 [cited 2013 Aug 2013].
[40] *Weka*. 2013 [cited 2014 Jan 2014]; Available from: http://www.cs.waikato.ac.nz/ml/weka/.
[41] James, G., Witten, D., Hastie, T., Tibshirani, R. , *An Introduction to Statistical Learning with Applications in R*. 2013: Springer, Inc.
[42] Dunteman, G., *Principal Component Analysis.* 1989: SAGE.
[43] Chok, N.S., *PEARSON'S VERSUS SPEARMAN'S AND KENDALL'S CORRELATION COEFFICIENTS FOR CONTINUOUS DATA*, in *Department of Biostatistics*. 2008, UNIVERSITY OF PITTSBURGH.
[44] Lee, B.C. and D.M. Brooks, *Accurate and efficient regression modeling for microarchitectural performance and power prediction.* SIGPLAN Not., 2006. 41(11): p. 185-194.
[45] Witten, I.H. and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. 2005: Morgan Kaufmann Publishers Inc.
[46] Hosmer, D.W. and S. Lemeshow, *Applied logistic regression (Wiley Series in probability and statistics)*. 2 ed. 2000: Wiley-Interscience Publication.
[47] Menard, S., *Applied Logistic Regression Analysis.* 1995: SAGE Publications.
[48] Ernest S. Shtatland, S.L.M., and Mary B. Barton., *Why We Need an R-Square Measure of Fit (and Not Only One).* in *Proceedings of the 25th SUGI.* 2000.
[49] Steyerberg, E.W., et al., *Internal validation of predictive models: Efficiency of some procedures for logistic regression analysis.* Journal of Clinical Epidemiology, 2001. 54(8): p. 774-781.
[50] Kohavi, R., *A study of cross-validation and bootstrap for accuracy estimation and model selection*, in *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 2*. 1995, Morgan Kaufmann Publishers Inc.: Montreal, Quebec, Canada. p. 1137-1143.
[51] Myrtveit, I., E. Stensrud, and M. Shepperd, *Reliability and validity in comparative studies of software prediction models.* Software Engineering, IEEE Transactions on, 2005. 31(5): p. 380-391.
[52] Shatnawi, R., et al., *Finding software metrics threshold values using ROC curves.* J. Softw. Maint. Evol., 2010. 22(1): p. 1-16.
[53] Elish, M.O.a. and A.-R. Al-Khiaty, *A suite of metrics for quantifying historical changes to predict future change-prone classes in object-oriented software.* J. Softw. Evol. and Proc, 2013.
[54] Al Dallal, J., *Object-oriented class maintainability prediction using internal quality attributes.* Information and Software Technology, 2013. 55(11): p. 2028-2048.

APPENDECIES

Appendix A: Kendall's tau correlation coefficients

Table A.1. Kendall's tau correlation coefficient with fault-proneness for Synapse, Velocity and Poi systems

	Synapse		Velocity		Poi	
Metric	Correlation coefficient	p-Value	Correlation coefficient	p-Value	Correlation coefficient	p-Value
PSRV	**0.366704**	0.000000	**0.239337**	0.000000	**0.424551**	0.000000
PSRVD	**-0.243160**	0.000000	**-0.196846**	0.000009	**-0.080418**	0.011823
PNCRV	**0.242528**	0.000000	**0.223927**	0.000000	**0.300086**	0.000000
PNCRVD	**-0.184574**	0.000011	-0.041027	0.355372	**-0.072810**	0.022654
PPCICRV	**0.205707**	0.000001	0.077496	0.080855	**0.198265**	0.000000
PPCICRVD	**0.177700**	0.000023	0.071117	0.109148	0.026583	0.405319
PFCRV	**0.349869**	0.000000	**0.177082**	0.000066	**0.202099**	0.000000
PFCRVD	**0.232034**	0.000000	**0.102149**	0.021386	**0.145227**	0.000005
PMCRV	**0.110449**	0.008469	**0.112813**	0.011043	**0.078168**	0.014407
PMCRVD	**0.110381**	0.008509	**0.107744**	0.015219	**0.075846**	0.017584
PDSCRV	**0.153171**	0.000261	0.014200	0.749064	**0.416215**	0.000000
PDSCRVD	**0.136193**	0.001169	-0.016325	0.713051	**0.241948**	0.000000
PExpCRV	**0.274842**	0.000000	**0.183421**	0.000036	**0.443775**	0.000000
PExpCRVD	**0.190590**	0.000006	**0.130522**	0.003280	**0.254767**	0.000000
PExcCRV	0.076250	0.069130	**0.160305**	0.000305	0.050478	0.114075
PExcCRVD	0.076699	0.067509	**0.159952**	0.000314	0.050449	0.114282
PTCRV	**0.157453**	0.000175	0.085823	0.053197	-0.013088	0.682024
PTCRVD	**0.156354**	0.000194	0.084058	0.058284	-0.015623	0.624803
PConCRV						
PConCRVD						
PComCRV	**0.198288**	0.000002	**0.243835**	0.000000	0.045152	0.157531
PComCRVD	-0.080028	0.056439	**0.115337**	0.009372	-0.054905	0.085664
PSCV	**0.353138**	0.000000	**0.245528**	0.000000	**0.444909**	0.000000
PSCVD	**-0.265011**	0.000000	**-0.189064**	0.000021	**-0.112344**	0.000437

Table A.2. Kendall's tau correlation coefficient with fault-proneness for Xalan, Camel and Ant systems

	Xalan		Camel		Ant	
Metric	Correlation coefficient	p-Value	Correlation coefficient	p-Value	Correlation coefficient	p-Value
PSRV	**0.247101**	0.000000	**0.169218**	0.000000	**0.371114**	0.000000
PSRVD	**-0.239990**	0.000000	0.010271	0.638562	**-0.203878**	0.000000
PNCRV	**0.218352**	0.000000	**0.125984**	0.000000	**0.316391**	0.000000
PNCRVD	**-0.117567**	0.000000	**0.067068**	0.002161	**-0.141656**	0.000000
PPCICRV	**0.247777**	0.000000	**0.046026**	0.035303	**0.236285**	0.000000
PPCICRVD	**0.209035**	0.000000	**0.044043**	0.043993	**0.210243**	0.000000
PFCRV	**0.106412**	0.000002	**0.144176**	0.000000	**0.323758**	0.000000
PFCRVD	0.018776	0.405730	**0.117169**	0.000000	**0.083757**	0.000645
PMCRV	**0.133583**	0.000000	**0.078351**	0.000339	**0.293858**	0.000000
PMCRVD	**0.115764**	0.000000	**0.077260**	0.000410	**0.274082**	0.000000
PDSCRV	**0.228158**	0.000000	0.040355	0.064966	**0.206078**	0.000000
PDSCRVD	**0.056094**	0.012993	0.028871	0.186726	**0.122313**	0.000001
PExpCRV	**0.201531**	0.000000	**0.160793**	0.000000	**0.352171**	0.000000
PExpCRVD	**0.076612**	0.000692	**0.140446**	0.000000	**0.161023**	0.000000
PExcCRV	**-0.051489**	0.022606	**0.065425**	0.002772	**0.155476**	0.000000
PExcCRVD	**-0.050787**	0.024515	**0.065425**	0.002772	**0.153464**	0.000000
PTCRV	**0.100192**	0.000009	0.005373	0.805907	**0.254173**	0.000000
PTCRVD	**0.064928**	0.004039	0.005361	0.806318	**0.190496**	0.000000
PConCRV	**0.050856**	0.024321			-0.019752	0.421051
PConCRVD	**0.050842**	0.024361			-0.019752	0.421051
PComCRV	**0.140569**	0.000000	**0.127551**	0.000000	**0.270971**	0.000000
PComCRVD	**-0.091197**	0.000054	**0.067594**	0.001993	-0.036734	0.134550
PSCV	**0.260134**	0.000000	**0.175839**	0.000000	**0.362338**	0.000000
PSCVD	**-0.237387**	0.000000	-0.002171	0.920908	**-0.226799**	0.000000

Table A.3. Kendall's tau correlation coefficient with fault density for Synapse, Velocity and Poi systems

	Synapse		Velocity		Poi	
Metric	Correlation coefficient	p-Value	Correlation coefficient	p-Value	Correlation coefficient	p-Value
PSRV	**0.257672**	0.000000	**0.144443**	0.001138	**0.153404**	0.000002
PSRVD	**-0.134579**	0.001337	**-0.104421**	0.018659	**0.131234**	0.000040
PNCRV	**0.157377**	0.000176	**0.163095**	0.000239	**0.120759**	0.000157
PNCRVD	**-0.103919**	0.013245	0.035763	0.420452	**0.111036**	0.000509
PPCICRV	**0.142322**	0.000693	0.030990	0.485115	0.053849	0.091860
PPCICRVD	**0.127900**	0.002298	0.029740	0.502891	0.039169	0.220144
PFCRV	**0.266410**	0.000000	**0.092090**	0.038033	0.032239	0.312878
PFCRVD	**0.220593**	0.000000	0.058862	0.184846	**0.064971**	0.041967
PMCRV	0.079940	0.056711	0.046883	0.290908	-0.010739	0.736731
PMCRVD	0.080168	0.056009	0.045221	0.308346	-0.010030	0.753546
PDSCRV	**0.095184**	0.023274	-0.035855	0.419265	**0.228199**	0.000000
PDSCRVD	**0.092429**	0.027579	-0.048355	0.276027	**0.300854**	0.000000
PExpCRV	**0.187313**	0.000008	**0.112804**	0.011050	**0.204953**	0.000000
PExpCRVD	**0.157165**	0.000179	**0.094798**	0.032720	**0.291769**	0.000000
PExcCRV	0.035990	0.390947	**0.114729**	0.009752	0.024444	0.444154
PExcCRVD	0.036724	0.381368	**0.114792**	0.009712	0.024547	0.442238
PTCRV	**0.164130**	0.000091	0.080102	0.071163	0.022179	0.487510
PTCRVD	**0.165129**	0.000083	0.080395	0.070132	0.020884	0.513269
PConCRV						
PConCRVD						
PComCRV	**0.131896**	0.001667	**0.164959**	0.000202	-0.060134	0.059781
PComCRVD	-0.022362	0.594002	**0.119110**	0.007293	-0.047136	0.140067
PSCV	**0.254422**	0.000000	**0.155685**	0.000453	**0.176426**	0.000000
PSCVD	**-0.148052**	0.000417	**-0.089421**	0.043971	**0.133767**	0.000028

Table A.4. Kendall's tau correlation coefficient with fault density for Xalan, Camel and Ant systems

	Xalan		Camel		Ant	
Metric	Correlation coefficient	p-Value	Correlation coefficient	p-Value	Correlation coefficient	p-Value
PSRV	**0.084008**	0.000199	**0.132027**	0.000000	**0.316884**	0.000000
PSRVD	**-0.044806**	0.047246	0.032241	0.140364	**-0.156320**	0.000000
PNCRV	**0.136088**	0.000000	**0.096612**	0.000010	**0.279451**	0.000000
PNCRVD	**0.063552**	0.004889	**0.070150**	0.001336	**-0.098702**	0.000058
PPCICRV	**0.089777**	0.000070	0.033546	0.125000	**0.201171**	0.000000
PPCICRVD	**0.089437**	0.000075	0.032204	0.140814	**0.182051**	0.000000
PFCRV	0.023935	0.289186	**0.116460**	0.000000	**0.274626**	0.000000
PFCRVD	0.007503	0.739691	**0.103135**	0.000002	**0.086830**	0.000405
PMCRV	0.035264	0.118387	**0.052331**	0.016703	**0.250093**	0.000000
PMCRVD	0.035217	0.118881	**0.052075**	0.017243	**0.235138**	0.000000
PDSCRV	**0.086844**	0.000120	0.028900	0.186279	**0.178088**	0.000000
PDSCRVD	**0.058372**	0.009742	0.020929	0.338497	**0.110101**	0.000007
PExpCRV	**0.055291**	0.014349	**0.122679**	0.000000	**0.301379**	0.000000
PExpCRVD	**0.056851**	0.011820	**0.110077**	0.000000	**0.153043**	0.000000
PExcCRV	**-0.057584**	0.010775	**0.072301**	0.000945	**0.135669**	0.000000
PExcCRVD	**-0.056396**	0.012513	**0.072301**	0.000945	**0.133813**	0.000000
PTCRV	0.041881	0.063658	0.001412	0.948500	**0.215379**	0.000000
PTCRVD	0.034433	0.127324	0.001437	0.947605	**0.165894**	0.000000
PConCRV	0.030255	0.180330			-0.018473	0.451745
PConCRVD	0.030292	0.179798			-0.018473	0.451745
PComCRV	0.020260	0.369645	**0.097520**	0.000008	**0.242908**	0.000000
PComCRVD	-0.031194	0.167182	**0.059803**	0.006240	-0.003318	0.892473
PSCV	**0.090050**	0.000067	**0.138910**	0.000000	**0.310624**	0.000000
PSCVD	-0.030639	0.174865	0.022759	0.297962	**-0.175206**	0.000000

Appendix B: Univariate Analysis Results with Respect to Fault Proneness and Faults Density

Table B.1 Univariate analysis result for Synapse classes with respect to fault proneness

Metric	C0	C1	-2log likelihood	Cox & snell R^2	Nagelkerke R^2	p-Value	P %	AUC
PSRV	2.5818	-0.2201	281.532	0.162	0.225	0.000000	69.5313	0.739
PSRVD	-0.0059	3.9273	313.537	0.051	0.070	0.000001	65.2344	0.671
PNCRV	1.795	-0.0192	310.764	0.061	0.084	0.000047	69.1406	0.612
PNCRVD	0.1889	0.3888	311.259	0.059	0.082	0.000308	66.4063	0.632
PPCICRV	0.8701	-0.0822	315.993	0.041	0.057	0.000855	68.3594	0.537
PPCICRVD	0.659	0.478	326.337	0.002	0.003	0.003055	66.4063	0.419
PFCRV	1.4708	-0.0539	291.806	0.128	0.177	0.000000	69.9219	0.665
PFCRVD	0.9373	-1.3283	317.996	0.034	0.047	0.000025	66.7969	0.637
PMCRV	0.7111	-0.0905	323.898	0.011	0.016	0.077743	67.1875	0.495
PMCRVD	0.7089	-14.0011	323.606	0.012	0.017	0.077052	66.7969	0.495
PDSCRV	0.8066	-0.031	322.497	0.017	0.023	0.013483	65.625	0.522
PDSCRVD	0.6935	-0.2319	326.732	0.000	0.000	0.022987	66.4063	0.518
PExpCRV	1.1711	-0.0753	307.766	0.072	0.099	0.000004	68.3594	0.632
PExpCRVD	0.8401	-2.1014	323.004	0.015	0.020	0.000645	65.2344	0.6
PExcCRV	0.6991	-0.0278	325.431	0.005	0.007	0.224062	66.4063	0.486
PExcCRVD	0.7021	-5.946	324.709	0.008	0.011	0.220434	66.7969	0.486
PTCRV	0.7419	-0.0736	320.882	0.023	0.032	0.011647	67.9688	0.504
PTCRVD	0.7335	-10.5918	320.511	0.024	0.034	0.011661	67.1875	0.504
PConCRV								
PConCRVD								
PComCRV	1.6094	-0.0115	315.819	0.042	0.058	0.001429	66.4063	0.558
PComCRVD	0.3378	0.215	316.807	0.038	0.053	0.127193	66.4063	0.553
PSCV	2.8437	-0.0663	285.076	0.150	0.209	0.000000	71.4844	0.726
PSCVD	-0.0201	1.0058	311.429	0.058	0.081	0.000000	64.8438	0.685

Table B.2 Univariate analysis result for Velocity classes with respect to fault proneness

Metric	C0	C1	-2 log likelihood	Cox & snell R^2	Nagelkerke R^2	p-Value	P %	AUC
PSRV	-1.4631	0.0941	279.281	0.061	0.085	0.000024	64.1921	0.649
PSRVD	0.0947	-4.8779	279.718	0.060	0.082	0.000257	61.1354	0.638
PNCRV	-1.6179	0.017	280.365	0.057	0.079	0.000359	63.3188	0.585
PNCRVD	-0.3674	-0.2377	288.056	0.025	0.034	0.456447	65.9389	0.527
PPCICRV	-0.7043	0.0396	292.511	0.006	0.008	0.240968	65.9389	0.498
PPCICRVD	-0.6556	-0.7432	293.756	0.000	0.000	0.275422	65.9389	0.444
PFCRV	-0.9579	0.0236	285.967	0.034	0.046	0.005547	66.8122	0.561
PFCRVD	-0.6134	-0.2997	293.212	0.002	0.003	0.089709	65.9389	0.42
PMCRV	-0.7287	0.0423	291.02	0.012	0.017	0.088518	65.5022	0.501
PMCRVD	-0.6686	0.6641	293.696	0.000	0.001	0.097926	65.5022	0.477
PDSCRV	-0.6948	0.0049	293.574	0.001	0.001	0.826758	65.9389	0.436
PDSCRVD	-0.5721	-0.9456	291.021	0.012	0.017	0.792726	65.9389	0.504
PExpCRV	-0.9428	0.0428	285.335	0.036	0.05	0.003540	64.1921	0.553
PExpCRVD	-0.7243	0.9727	293.118	0.003	0.004	0.030203	65.5022	0.505
PExcCRV	-0.6998	0.5111	287.24	0.028	0.039	0.015169	67.2489	0.507
PExcCRVD	-0.6998	255.3559	287.24	0.028	0.039	0.015172	67.2489	0.507
PTCRV	-0.7071	0.0283	292.176	0.007	0.01	0.195661	64.1921	0.503
PTCRVD	-0.6829	1.3996	293.195	0.003	0.004	0.198604	65.5022	0.506
PConCRV								
PConCRVD								
PComCRV	-1.3731	0.0112	279.631	0.060	0.083	0.000194	65.9389	0.57
PComCRVD	-0.6505	-0.0102	293.766	0.000	0.000	0.044153	65.5022	0.417
PSCV	-1.6953	0.0337	277.703	0.068	0.094	0.000019	63.7555	0.646
PSCVD	0.0582	-1.1882	279.27	0.061	0.085	0.000451	62.0087	0.634

Table B.3 Univariate analysis result for Poi classes with respect to fault proneness

Metric	C0	C1	-2 log likelihood	Cox & snell R^2	Nagelkerke R^2	p-Value	P %	AUC
PSRV	-1.6893	0.2202	456.642	0.232	0.318	0.000000	74.0319	0.781
PSRVD	0.9099	-2.1586	567.201	0.012	0.016	0.040126	63.7813	0.55
PNCRV	-0.7413	0.0231	525.379	0.102	0.14	0.000000	69.7039	0.638
PNCRVD	0.8636	-0.2976	560.361	0.027	0.037	0.064584	65.6036	0.549
PPCICRV	0.1261	0.0549	549.122	0.052	0.071	0.000011	64.0091	0.574
PPCICRVD	0.6857	-0.914	570.112	0.005	0.007	0.524992	64.0091	0.461
PFCRV	0.2816	0.0283	554.043	0.041	0.056	0.000011	64.0091	0.563
PFCRVD	0.5612	0.1283	572.412	0.000	0.000	0.000771	64.0091	0.481
PMCRV	0.539	0.0454	572.412	0.000	0.000	0.101915	64.0091	0.504
PMCRVD	0.5691	1.2413	572.375	0.000	0.000	0.108452	64.0091	0.499
PDSCRV	-0.4092	0.0756	498.078	0.156	0.214	0.000000	75.1708	0.704
PDSCRVD	0.3264	1.5455	564.253	0.019	0.026	0.000000	64.0091	0.647
PExpCRV	-0.6453	0.1596	475.096	0.199	0.273	0.000000	76.082	0.715
PExpCRVD	0.1851	4.3931	556.119	0.037	0.05	0.000000	64.0091	0.666
PExcCRV	0.5686	0.6352	570.718	0.004	0.006	0.291296	64.0091	0.491
PExcCRVD	0.5686	168.1293	570.718	0.004	0.006	0.291297	64.0091	0.491
PTCRV	0.5773	-0.0014	572.485	0.000	0.000	0.784279	63.7813	0.466
PTCRVD	0.6056	-1.0251	566.684	0.013	0.018	0.741730	64.6925	0.493
PConCRV	0.5758		572.494	0.000	0.000		64.0091	0.493
PConCRVD								
PComCRV								
PComCRVD	0.8349	-0.4386	557.242	0.034	0.047	0.196166	68.3371	0.526
PSCV	-2.0946	0.0695	452.52	0.239	0.328	0.000000	73.3485	0.781
PSCVD	1.0356	-0.7788	562.096	0.023	0.032	0.004043	64.9203	0.575

Table B.4 Univariate analysis result for Xalan classes with respect to fault proneness

Metric	C0	C1	-2 log likelihood	Cox & snell R^2	Nagelkerke R^2	p-Value	P %	AUC
PSRV	-1.0345	0.0872	1136.373	0.080	0.107	0.000000	60.2286	0.66
PSRVD	0.578	-5.3465	1137.865	0.079	0.105	0.000000	64.2286	0.666
PNCRV	-1.2155	0.0245	1166.477	0.048	0.064	0.000000	54.4	0.566
PNCRVD	0.1581	-0.343	1172.902	0.041	0.055	0.000024	61.1429	0.588
PPCICRV	-0.3625	0.0796	1160.694	0.055	0.073	0.000000	62.1714	0.572
PPCICRVD	-0.1354	0.5838	1209.233	0.001	0.001	0.000000	51.5429	0.527
PFCRV	-0.2963	0.0127	1198.573	0.013	0.017	0.001046	55.8857	0.543
PFCRVD	-0.013	-0.907	1200.597	0.010	0.014	0.536903	52.1143	0.485
PMCRV	-0.2368	0.0386	1191.729	0.020	0.027	0.000070	56.8	0.529
PMCRVD	-0.1237	0.1247	1209.765	0.000	0.000	0.000376	52.6857	0.495
PDSCRV	-0.6045	0.0303	1155.898	0.060	0.080	0.000000	58.6286	0.605
PDSCRVD	-0.0168	-0.7049	1201.765	0.009	0.012	0.056516	53.0286	0.461
PExpCRV	-0.443	0.0363	1178.589	0.035	0.047	0.000000	56.6857	0.594
PExpCRVD	-0.0159	-1.2093	1198.872	0.012	0.017	0.010016	51.6571	0.455
PExcCRV	-0.1019	-0.0118	1207.42	0.003	0.004	0.128035	53.0286	0.503
PExcCRVD	-0.1098	-1.451	1208.651	0.001	0.002	0.131007	53.0286	0.503
PTCRV	-0.2298	0.0198	1200.047	0.011	0.015	0.002913	55.7714	0.518
PTCRVD	-0.069	-0.7638	1203.921	0.007	0.009	0.042842	53.0286	0.462
PConCRV	-0.1262	0.9847	1206.768	0.003	0.005	0.132794	53.2571	0.497
PConCRVD	-0.1262	111.936	1206.768	0.003	0.005	0.132794	53.2571	0.497
PComCRV	-0.5455	0.0062	1192.331	0.020	0.026	0.000030	55.4286	0.531
PComCRVD	0.233	-0.4387	1155.213	0.060	0.081	0.001463	58.1714	0.558
PSCV	-1.3363	0.0316	1125.458	0.092	0.123	0.000000	65.4857	0.664
PSCVD	0.5338	-1.2553	1138.379	0.078	0.105	0.000000	62.5143	0.665

Table B.5 Univariate analysis result for Camel classes with respect to fault proneness

Metric	C0	C1	-2 log likelihood	Cox & snell R^2	Nagelkerke R^2	p-Value	P %	AUC
PSRV	2.1188	-0.1032	898.055	0.039	0.061	0.000000	79.8499	0.63
PSRVD	1.4646	-0.4964	934.296	0.001	0.001	0.703701	79.9571	0.46
PNCRV	1.8013	-0.0085	918.6	0.017	0.027	0.000044	79.9571	0.577
PNCRVD	1.4155	-0.0294	934.566	0.000	0.000	0.016583	79.9571	0.488
PPCICRV	1.4306	-0.0427	930.405	0.005	0.008	0.159056	80.0643	0.498
PPCICRVD	1.3808	0.2298	934.822	0.000	0.000	0.171715	79.9571	0.471
PFCRV	1.6262	-0.0192	917.366	0.019	0.029	0.000005	79.8499	0.548
PFCRVD	1.5112	-0.6401	926.17	0.009	0.015	0.000090	80.1715	0.555
PMCRV	1.4243	-0.0445	929.025	0.006	0.010	0.016622	80.0643	0.497
PMCRVD	1.3981	-1.7539	933.534	0.001	0.002	0.017281	79.8499	0.498
PDSCRV	1.4361	-0.0119	933.38	0.002	0.003	0.218141	79.9571	0.5
PDSCRVD	1.3526	0.3687	933.735	0.001	0.002	0.353691	79.9571	0.47
PExpCRV	1.6115	-0.0587	909.03	0.027	0.043	0.000000	80.2787	0.555
PExpCRVD	1.4708	-1.875	928.158	0.007	0.011	0.000005	79.8499	0.559
PExcCRV	1.389	-1.4626	931.64	0.003	0.005	0.045732	79.9571	0.493
PExcCRVD	1.389	-137.985	931.64	0.003	0.005	0.045732	79.9571	0.493
PTCRV	1.3849	-0.0053	934.833	0.000	0.000	0.869814	79.9571	0.481
PTCRVD	1.3884	-1.0373	934.032	0.001	0.001	0.869816	79.8499	0.487
PConCRV								
PConCRVD								
PComCRV	1.7605	-0.0066	919.44	0.016	0.026	0.000094	79.9571	0.552
PComCRVD	1.4628	-0.0702	932.381	0.003	0.004	0.018674	79.9571	0.531
PSCV	2.2528	-0.0336	898.318	0.038	0.061	0.000000	79.4212	0.631
PSCVD	1.428	-0.0693	934.669	0.000	0.000	0.935918	79.9571	0.441

Table B.6 Univariate analysis result for Ant classes with respect to fault proneness

Metric	C0	C1	-2 log likelihood	Cox & snell R^2	Nagelkerke R^2	p-Value	P %	AUC
PSRV	-3.3429	0.1775	627.267	0.195	0.298	0.000000	81.2416	0.793
PSRVD	-0.4191	-6.9854	743.702	0.058	0.089	0.000000	77.5978	0.67
PNCRV	-3.1173	0.0268	699.161	0.113	0.173	0.000000	77.5978	0.682
PNCRVD	-0.787	-0.5412	757.021	0.041	0.063	0.000003	77.5978	0.617
PPCICRV	-1.4731	0.0958	752.547	0.047	0.072	0.000000	77.8677	0.561
PPCICRVD	-1.2581	0.9783	787.911	0.001	0.001	0.000000	77.5978	0.56
PFCRV	-2.2326	0.044	700.331	0.112	0.171	0.000000	76.1134	0.699
PFCRVD	-1.1382	-0.5881	785.294	0.004	0.006	0.008978	77.5978	0.426
PMCRV	-1.4914	0.0936	734.717	0.070	0.107	0.000000	79.4872	0.564
PMCRVD	-1.2688	1.8298	786.478	0.003	0.004	0.000000	77.3279	0.573
PDSCRV	-1.5782	0.0466	756.907	0.042	0.063	0.000000	78.1377	0.576
PDSCRVD	-1.1626	-1.1639	782.55	0.008	0.012	0.000337	77.5978	0.411
PExpCRV	-2.1487	0.0899	683.86	0.132	0.201	0.000000	78.9474	0.717
PExpCRVD	-1.2227	-0.2511	788.186	0.000	0.000	0.000001	77.5978	0.389
PExcCRV	-1.3005	0.0381	774.055	0.019	0.029	0.000021	78.2726	0.511
PExcCRVD	-1.2493	0.6303	787.771	0.001	0.001	0.000025	77.4629	0.518
PTCRV	-1.6328	0.0483	745.441	0.056	0.086	0.000000	77.4629	0.594
PTCRVD	-1.2205	-0.3866	787.826	0.001	0.001	0.000000	77.5978	0.378
PConCRV	-1.2406	-0.9322	787.844	0.001	0.001	0.591398	77.5978	0.491
PConCRVD	-1.2406	-194.189	787.844	0.001	0.001	0.591398	77.5978	0.491
PComCRV	-3.0155	0.0215	721.045	0.087	0.133	0.000000	77.5978	0.624
PComCRVD	-0.9495	-0.3315	771.762	0.022	0.034	0.236590	77.5978	0.528
PSCV	-3.7251	0.0619	638.888	0.183	0.279	0.000000	80.4318	0.779
PSCVD	-0.3466	-2.0975	728.669	0.077	0.118	0.000000	77.5978	0.69

Table B.7 Univariate analysis result for Synapse classes with respect to fault density

Metric	C0	C1	R^2	Adjusted R^2	p-Value	MAE	RMSE	AE Std
PSRV	6.80	0.27	0.003	-0.001	0.000000	12.40	23.01	19.42
PSRVD	3.03	30.86	0.051	0.047	0.001505	13.31	23.29	19.15
PNCRV	9.85	-0.01	0.000	-0.004	0.004353	12.58	22.97	19.26
PNCRVD	7.81	0.80	0.004	0.001	0.013001	12.76	23.06	19.26
PPCICRV	8.25	0.37	0.007	0.003	0.012287	12.46	22.96	19.32
PPCICRVD	8.29	14.48	0.017	0.013	0.023189	12.75	23.43	19.69
PFCRV	7.70	0.10	0.004	0.001	0.000001	12.34	23.00	19.44
PFCRVD	4.30	25.95	0.111	0.107	0.000058	11.28	22.69	19.73
PMCRV	8.98	0.13	0.000	-0.004	0.165574	12.58	22.98	19.27
PMCRVD	8.90	52.03	0.003	-0.001	0.163066	12.60	23.08	19.37
PDSCRV	9.42	-0.11	0.002	-0.002	0.094532	12.58	22.94	19.23
PDSCRVD	9.00	0.39	0.000	-0.004	0.105930	12.57	22.99	19.29
PExpCRV	9.24	-0.04	0.000	-0.004	0.000573	12.55	22.96	19.27
PExpCRVD	6.00	41.99	0.048	0.045	0.004363	12.14	23.36	19.99
PExcCRV	9.08	-0.09	0.000	-0.003	0.533081	12.54	22.93	19.23
PExcCRVD	9.06	-9.99	0.000	-0.004	0.527917	12.58	22.95	19.23
PTCRV	7.84	1.51	0.083	0.079	0.004177	12.48	23.49	19.94
PTCRVD	7.67	113.26	0.350	0.347	0.003973	11.76	21.84	18.45
PConCRV								
PConCRVD								
PComCRV	13.23	-0.05	0.010	0.006	0.021717	12.89	23.02	19.11
PComCRVD	9.56	-0.27	0.002	-0.002	0.334625	12.55	22.96	19.26
PSCV	4.74	0.14	0.008	0.004	0.000000	12.31	23.02	19.49
PSCVD	3.51	6.99	0.049	0.045	0.000429	13.32	23.24	19.08

Table B.8 Univariate analysis result for Velocity classes with respect to fault density

Metric	C0	C1	R^2	Adjusted R^2	p-Value	MAE	RMSE	AE Std
PSRV	15.61	-0.42	0.007	0.003	0.002089	17.34	29.42	23.77
PSRVD	11.34	4.76	0.001	-0.004	0.017968	17.30	29.55	23.96
PNCRV	11.71	0.01	0.000	-0.004	0.003838	17.18	29.57	24.07
PNCRVD	10.94	0.88	0.003	-0.001	0.769255	17.30	29.56	23.97
PPCICRV	12.66	-0.49	0.004	0.000	0.607429	17.27	29.42	23.82
PPCICRVD	12.66	-0.49	0.003	-0.001	0.638388	17.27	29.42	23.82
PFCRV	14.22	-0.18	0.010	0.005	0.108865	17.18	29.34	23.78
PFCRVD	12.77	-3.77	0.002	-0.002	0.328084	17.18	29.45	23.92
PMCRV	12.66	-0.34	0.004	-0.001	0.442294	17.08	29.42	23.95
PMCRVD	12.31	-13.67	0.001	-0.004	0.451943	17.27	29.57	24.01
PDSCRV	13.91	-0.26	0.012	0.008	0.561520	17.07	29.33	23.85
PDSCRVD	12.44	-2.63	0.001	-0.004	0.413409	17.27	29.66	24.11
PExpCRV	13.77	-0.26	0.007	0.003	0.045830	17.38	29.42	23.74
PExpCRVD	12.91	-11.96	0.002	-0.002	0.108864	17.30	29.51	23.91
PExcCRV	12.14	0.01	0.000	-0.004	0.059247	17.22	29.48	23.93
PExcCRVD	12.12	4.69	0.000	-0.004	0.059001	17.18	29.47	23.95
PTCRV	11.79	0.24	0.002	-0.002	0.188726	17.21	29.54	24.01
PTCRVD	11.73	27.92	0.005	0.001	0.182990	17.25	29.70	24.18
PConCRV								
PConCRVD								
PComCRV	14.18	-0.03	0.003	-0.001	0.006454	17.39	29.52	23.86
PComCRVD	11.60	0.55	0.001	-0.004	0.061529	17.23	29.61	24.08
PSCV	15.36	-0.11	0.004	0.000	0.001545	17.31	29.45	23.83
PSCVD	10.60	2.31	0.002	-0.002	0.033795	17.34	29.54	23.92

Table B.9 Univariate analysis result for Poi classes with respect to fault density

Metric	C0	C1	R^2	Adjusted R^2	p-Value	MAE	RMSE	AE Std
PSRV	15.63	-0.19	0.002	0.000	0.000001	13.34	23.01	18.74
PSRVD	6.71	44.61	0.040	0.038	0.003165	12.64	22.62	18.76
PNCRV	15.45	-0.03	0.002	0.000	0.001465	13.33	22.99	18.74
PNCRVD	11.69	1.91	0.010	0.008	0.077867	13.15	22.89	18.73
PPCICRV	15.43	-0.22	0.008	0.005	0.167480	13.33	22.85	18.56
PPCICRVD	14.06	-4.72	0.001	-0.001	0.619603	13.36	22.98	18.70
PFCRV	14.36	-0.07	0.002	0.000	0.389541	13.18	22.95	18.79
PFCRVD	12.95	4.91	0.002	0.000	0.197342	13.35	23.02	18.75
PMCRV	13.78	-0.28	0.003	0.000	0.793238	13.17	22.88	18.71
PMCRVD	13.72	-38.67	0.003	0.000	0.787013	13.18	22.88	18.71
PDSCRV	13.21	0.02	0.000	-0.002	0.000000	13.24	23.00	18.81
PDSCRVD	10.51	17.39	0.026	0.023	0.000000	12.56	22.85	19.08
PExpCRV	14.09	-0.07	0.000	-0.002	0.000000	13.32	23.00	18.75
PExpCRVD	12.04	15.26	0.005	0.002	0.000000	12.90	22.92	18.95
PExcCRV	13.51	-0.02	0.000	-0.002	0.550716	13.28	22.94	18.70
PExcCRVD	13.49	9.84	0.000	-0.002	0.548889	13.33	23.03	18.78
PTCRV	13.28	0.21	0.002	0.000	0.587000	13.25	22.97	18.77
PTCRVD	13.60	-2.64	0.001	-0.001	0.618051	13.23	22.91	18.71
PConCRV								
PConCRVD								
PComCRV	16.39	-0.06	0.017	0.015	0.141729	13.13	22.77	18.60
PComCRVD	14.62	-1.97	0.006	0.003	0.086866	13.27	22.97	18.75
PSCV	16.43	-0.07	0.003	0.001	0.000000	13.41	22.99	18.68
PSCVD	7.24	10.82	0.039	0.036	0.007407	12.58	22.68	18.86

Table B.10 Univariate analysis result for Xalan classes with respect to fault density

Metric	C0	C1	R^2	Adjusted R^2	p-Value	MAE	RMSE	AE Std
PSRV	13.04	-0.46	0.028	0.027	0.000155	10.66	19.31	16.10
PSRVD	6.81	9.37	0.010	0.009	0.001780	10.56	19.65	16.57
PNCRV	10.40	-0.05	0.003	0.001	0.000004	10.58	19.56	16.45
PNCRVD	6.55	1.76	0.024	0.023	0.795760	10.50	19.36	16.26
PPCICRV	8.95	-0.24	0.006	0.005	0.002461	10.47	19.51	16.46
PPCICRVD	7.99	7.85	0.001	0.000	0.005126	10.54	19.59	16.51
PFCRV	10.22	-0.15	0.019	0.018	0.379029	10.61	19.39	16.23
PFCRVD	8.91	-5.90	0.005	0.004	0.859265	10.57	19.52	16.41
PMCRV	8.79	-0.20	0.006	0.005	0.242690	10.53	19.51	16.42
PMCRVD	8.18	-0.23	0.000	-0.001	0.280580	10.57	19.60	16.50
PDSCRV	10.60	-0.15	0.018	0.017	0.001694	10.61	19.41	16.25
PDSCRVD	8.25	-0.46	0.000	-0.001	0.200703	10.55	19.59	16.50
PExpCRV	10.50	-0.26	0.020	0.019	0.032806	10.49	19.38	16.29
PExpCRVD	8.29	-1.10	0.001	-0.001	0.149885	10.54	19.56	16.47
PExcCRV	8.37	-0.11	0.003	0.001	0.055696	10.51	19.54	16.47
PExcCRVD	8.25	-8.15	0.000	-0.001	0.059618	10.55	19.57	16.48
PTCRV	8.78	-0.11	0.004	0.002	0.160751	10.55	19.53	16.43
PTCRVD	7.96	3.08	0.001	0.000	0.333688	10.54	19.59	16.51
PConCRV	8.18	-0.04	0.000	-0.001	0.315115	10.55	19.57	16.48
PConCRVD	8.18	3.68	0.000	-0.001	0.314799	10.56	19.58	16.49
PComCRV	14.28	-0.09	0.045	0.044	0.501218	10.40	19.15	16.08
PComCRVD	9.02	-0.91	0.005	0.004	0.011714	10.52	19.52	16.44
PSCV	14.34	-0.16	0.030	0.029	0.000071	10.59	19.30	16.13
PSCVD	6.73	2.49	0.012	0.011	0.005167	10.57	19.76	16.69

Table B.11 Univariate analysis result for Camel classes with respect to fault density

Metric	C0	C1	R^2	Adjusted R^2	p-Value	MAE	RMSE	AE Std
PSRV	13.77	-0.30	0.001	0.000	0.000001	19.31	43.21	38.67
PSRVD	3.11	54.08	0.023	0.022	0.240208	18.89	42.86	38.49
PNCRV	13.50	-0.04	0.001	0.000	0.000811	19.27	43.19	38.67
PNCRVD	10.26	1.46	0.002	0.001	0.013735	19.11	43.23	38.79
PPCICRV	12.02	-0.22	0.000	-0.001	0.280417	19.15	43.19	38.73
PPCICRVD	11.93	-9.44	0.000	-0.001	0.291246	19.13	43.19	38.74
PFCRV	12.26	-0.04	0.000	-0.001	0.000103	19.21	43.23	38.75
PFCRVD	10.02	10.07	0.006	0.005	0.000423	18.90	43.20	38.87
PMCRV	12.04	-0.32	0.001	0.000	0.092418	19.19	43.17	38.69
PMCRVD	11.97	-22.14	0.001	0.000	0.094275	19.17	43.17	38.71
PDSCRV	12.58	-0.19	0.001	0.000	0.353873	19.21	43.18	38.69
PDSCRVD	12.26	-4.87	0.001	0.000	0.489871	19.13	43.18	38.74
PExpCRV	12.47	-0.20	0.001	0.000	0.000053	19.27	43.20	38.68
PExpCRVD	11.98	-4.09	0.000	-0.001	0.000231	19.19	43.21	38.74
PExcCRV	11.73	1.47	0.003	0.002	0.020224	19.08	43.18	38.76
PExcCRVD	11.73	138.44	0.003	0.002	0.020224	19.08	43.18	38.76
PTCRV	11.87	-0.22	0.000	-0.001	0.963870	19.15	43.19	38.74
PTCRVD	11.77	11.11	0.000	-0.001	0.962414	19.17	43.24	38.77
PConCRV								
PConCRVD								
PComCRV	13.95	-0.04	0.002	0.001	0.001713	19.28	43.21	38.69
PComCRVD	9.47	2.20	0.008	0.007	0.036729	18.98	43.23	38.86
PSCV	13.50	-0.07	0.001	0.000	0.000000	19.28	43.22	38.71
PSCVD	3.17	13.60	0.026	0.025	0.426001	18.93	42.83	38.44

Table B.12 Univariate analysis result for Ant classes with respect to fault density

Metric	C0	C1	R^2	Adjusted R^2	p-Value	MAE	RMSE	AE Std
PSRV	1.80	0.12	0.008	0.006	0.000000	4.46	9.51	8.41
PSRVD	3.21	-1.39	0.000	-0.001	0.000000	4.69	9.56	8.34
PNCRV	1.59	0.02	0.008	0.007	0.000000	4.52	9.51	8.37
PNCRVD	2.77	0.22	0.001	0.000	0.000122	4.76	9.60	8.34
PPCICRV	2.88	0.07	0.001	0.000	0.000000	4.67	9.54	8.33
PPCICRVD	3.05	-1.85	0.000	-0.001	0.000000	4.72	9.54	8.30
PFCRV	2.69	0.02	0.001	0.000	0.000000	4.64	9.54	8.34
PFCRVD	3.36	-1.79	0.003	0.002	0.011291	4.71	9.53	8.30
PMCRV	2.80	0.11	0.005	0.004	0.000000	4.61	9.52	8.33
PMCRVD	2.97	3.98	0.001	-0.001	0.000000	4.73	9.56	8.32
PDSCRV	2.75	0.04	0.002	0.001	0.000000	4.66	9.54	8.33
PDSCRVD	3.12	-1.16	0.001	0.000	0.000921	4.70	9.54	8.30
PExpCRV	2.42	0.07	0.006	0.005	0.000000	4.54	9.52	8.37
PExpCRVD	2.97	0.63	0.000	-0.001	0.000002	4.72	9.55	8.31
PExcCRV	2.94	0.07	0.003	0.002		4.67	9.54	8.32
PExcCRVD	2.98	3.68	0.001	0.000	0.000099	4.73	9.58	8.34
PTCRV	2.84	0.03	0.001	0.000	0.000000	4.66	9.54	8.33
PTCRVD	3.11	-1.57	0.001	0.000	0.000000	4.71	9.54	8.30
PConCRV	3.02	-0.09	0.000	-0.001	0.595338	4.72	9.54	3.24
PConCRVD	3.02	-18.90	0.000	-0.001	0.595338	4.72	9.54	8.30
PComCRV	2.21	0.01	0.003	0.002	0.000000	4.62	9.54	8.35
PComCRVD	3.09	-0.06	0.000	-0.001	0.526253	4.71	9.55	8.31
PSCV	1.74	0.04	0.006	0.005	0.000000	4.49	9.52	8.40
PSCVD	3.31	-0.55	0.001	-0.001	0.000000	4.68	9.56	8.35

Appendix C: Principle component analysis

Table C.1 PCA for Coding standard violations-based metrics (Synapse)

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9
PSRV	-0.241	**0.789**	0.251	0.115	0.154	0.268	0.309	0.183	0.053
PSRVD	**0.941**	-0.183	0.128	-0.023	-0.033	0.033	0.005	0.09	0.152
PNCRV	0.021	**0.875**	-0.038	-0.025	-0.019	0.017	0.034	-0.126	-0.129
PNCRVD	**0.866**	0.105	-0.129	-0.038	-0.014	-0.099	-0.167	-0.138	-0.233
PPCICRV	-0.078	0.268	0.057	0.106	0.208	0.027	0.02	**0.849**	0.002
PPCICRVD	0.125	-0.201	0.084	-0.018	-0.072	-0.042	0.004	**0.859**	-0.04
PFCRV	-0.319	**0.591**	0.565	0.11	-0.099	0.065	-0.065	-0.003	0.136
PFCRVD	0	0.149	**0.806**	0.009	-0.13	0.074	-0.071	-0.118	0.145
PMCRV	-0.04	0.088	0.02	-0.016	**0.937**	0.061	0.041	0.121	0.006
PMCRVD	-0.015	-0.004	0.012	-0.001	**0.934**	-0.005	0.04	-0.021	0.014
PDSCRV	-0.099	0.247	0.017	0.065	0.091	**0.887**	0.056	0.062	-0.029
PDSCRVD	0.068	0.005	0.025	-0.023	-0.031	**0.932**	-0.036	-0.077	-0.005
PExpCRV	-0.263	0.466	-0.015	-0.006	0.128	0.057	**0.761**	0.073	0.032
PExpCRVD	0.016	0.042	0.17	-0.009	0.005	-0.02	**0.938**	-0.031	0.004
PExcCRV	-0.033	0.06	-0.005	**0.98**	-0.009	0.035	-0.01	0.03	-0.004
PExcCRVD	-0.026	0.045	-0.007	**0.981**	-0.008	0.005	0.006	0.04	-0.001
PTCRV	0.005	0.103	**0.635**	-0.043	0.185	0.007	0.097	0.159	-0.12
PTCRVD	0.243	-0.098	**0.742**	-0.007	0.013	-0.055	0.288	0.153	-0.133
PComCRV	-0.396	0.385	0.027	0.024	0.04	-0.023	0.139	-0.076	**0.676**
PComCRVD	0.302	-0.219	-0.07	-0.022	0.001	-0.024	-0.056	0	**0.859**
PSCV	-0.243	**0.697**	0.307	0.173	0.163	0.304	0.318	0.186	0.091
PSCVD	**0.926**	-0.236	0.116	-0.017	-0.029	0.019	-0.012	0.096	0.158

Table C.2 PCA for Coding standard violations-based metrics (Velocity)

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8
PSRV	**0.858**	-0.103	0.147	0.126	0.085	0.251	0.261	0.199
PSRVD	-0.147	**0.938**	-0.01	-0.055	0.016	0.014	0.035	0.187
PNCRV	**0.691**	0.144	-0.132	-0.093	0.046	0.135	0.051	-0.244
PNCRVD	-0.051	**0.815**	-0.102	-0.118	-0.014	-0.011	-0.154	-0.147
PPCICRV	0.33	-0.099	0.089	0.032	0.014	**0.873**	0.004	0.028
PPCICRVD	0.121	0.013	0.015	0.057	-0.026	**0.918**	0.051	-0.029
PFCRV	**0.825**	-0.018	0.2	0.049	0.058	0.087	-0.079	0.162
PFCRVD	0.4	**0.696**	0.105	-0.017	0.003	-0.074	-0.133	0.072
PMCRV	0.204	-0.125	0.046	**0.871**	0.041	0.047	-0.067	0.07
PMCRVD	-0.018	0.015	0.002	**0.92**	-0.014	0.042	-0.013	0.02
PDSCRV	0.542	-0.1	0.043	0.127	0.001	0.063	0.082	**0.687**
PDSCRVD	-0.001	0.169	-0.047	0.021	0.007	-0.034	0.032	**0.834**
PExpCRV	0.581	-0.24	0.062	-0.06	-0.004	0.136	**0.609**	0.147
PExpCRVD	0.142	0.031	0.03	-0.109	-0.008	0.023	**0.902**	0.062
PExcCRV	0.12	-0.024	0.04	0.042	**0.942**	0.021	-0.012	-0.017
PExcCRVD	0.028	0.011	-0.027	-0.02	**0.944**	-0.032	-0.013	0.028
PTCRV	0.191	-0.066	**0.918**	0.045	0.044	0.121	0.007	0.017
PTCRVD	0.011	-0.007	**0.948**	-0.002	-0.029	-0.02	0.021	-0.035
PComCRV	**0.623**	-0.047	-0.123	0.277	-0.083	-0.029	0.491	-0.203
PComCRVD	0.179	**0.654**	-0.101	0.17	-0.086	-0.138	0.323	-0.256
PSCV	**0.813**	-0.052	0.18	0.191	0.132	0.236	0.284	0.188
PSCVD	-0.255	**0.887**	-0.004	-0.051	0.031	0.031	-0.001	0.226

Table C.3 PCA for Coding standard violations-based metrics (Poi)

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9
PSRV	-0.193	**0.694**	0.417	0.362	0.178	0.037	0.342	0.054	0.095
PSRVD	**0.963**	-0.003	0.075	0.124	-0.034	0.002	0.097	0.05	0.081
PNCRV	0.092	**0.708**	0.224	0.246	0.033	-0.022	0.123	-0.077	-0.075
PNCRVD	**0.828**	0.12	-0.062	-0.057	-0.047	-0.03	-0.152	-0.129	-0.038
PPCICRV	-0.09	0.246	0.153	**0.892**	0.027	-0.054	0.066	-0.067	-0.013
PPCICRVD	0.273	-0.164	-0.069	**0.851**	-0.031	-0.01	0.025	0.089	-0.104
PFCRV	-0.154	**0.854**	-0.044	-0.106	0.141	0.096	-0.016	0.171	0.06
PFCRVD	0.27	**0.692**	-0.236	-0.251	-0.015	0.017	-0.038	0.173	-0.071
PMCRV	-0.088	0.15	0.048	0.008	**0.939**	0.064	-0.001	0.016	0.047
PMCRVD	0.007	0.057	-0.007	0.003	**0.954**	-0.001	0.019	0.055	-0.016
PDSCRV	-0.203	0.294	**0.846**	0.119	0.067	0.005	0.162	-0.018	0.063
PDSCRVD	0.329	-0.131	**0.842**	-0.04	-0.045	0.004	0.066	-0.013	-0.118
PExpCRV	-0.275	0.348	0.228	0.068	0.061	0.011	**0.788**	-0.01	0.058
PExpCRVD	0.213	-0.035	0.05	0.032	-0.04	0.028	**0.925**	0.043	-0.093
PExcCRV	-0.016	0.062	0.005	-0.027	0.068	**0.963**	-0.013	0.008	-0.008
PExcCRVD	-0.002	0.02	0.007	-0.024	-0.007	**0.963**	0.049	0.02	-0.004
PTCRV	-0.032	0.1	0.056	-0.029	0.04	-0.019	0.013	0.035	**0.879**
PTCRVD	0.157	-0.114	-0.094	-0.058	-0.013	0.008	-0.055	-0.029	**0.858**
PComCRV	-0.243	0.334	0.085	-0.034	0.111	0.032	0.043	**0.834**	0.039
PComCRVD	0.18	-0.02	-0.078	0.045	-0.009	0.005	0.012	**0.942**	-0.02
PSCV	-0.152	**0.602**	0.425	0.344	0.222	0.038	0.394	0.12	0.067
PSCVD	**0.944**	-0.132	0.05	0.076	-0.019	0.005	0.067	0.082	0.104

Table C.4 PCA for Coding standard violations-based metrics (Xalan)

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10
PSRV	**0.952**	-0.055	0.122	0.025	0.143	0.113	-0.045	0.139	0.075	0.039
PSRVD	-0.12	**0.946**	-0.011	-0.002	0.012	0.054	0.135	0.053	0.148	0.176
PNCRV	0.479	-0.067	0.007	0.012	0.105	0.037	-0.086	0.085	-0.096	**0.708**
PNCRVD	-0.337	0.297	0.002	-0.01	-0.028	0.021	-0.111	-0.025	0.064	**0.748**
PPCICRV	0.372	-0.064	-0.079	-0.024	**0.809**	-0.04	-0.055	-0.085	-0.014	-0.021
PPCICRVD	-0.03	0.04	0	-0.008	**0.889**	-0.022	0.061	-0.002	0.189	0.059
PFCRV	0.598	-0.082	0.146	0.003	-0.055	0.068	-0.075	**0.666**	-0.021	-0.021
PFCRVD	0.036	0.081	-0.01	-0.017	-0.047	0.009	0.16	**0.918**	0.15	0.044
PMCRV	**0.562**	-0.019	0.148	0.086	0.123	0.023	-0.311	0.023	0.486	-0.127
PMCRVD	0.086	0.161	0.039	0.049	0.24	0.002	-0.011	0.201	**0.809**	-0.028
PDSCRV	**0.826**	-0.133	-0.051	0.043	-0.049	-0.014	0.184	-0.11	0.172	0.057
PDSCRVD	0.118	0.118	-0.068	-0.003	-0.152	-0.092	**0.602**	-0.114	0.572	0.13
PExpCRV	**0.809**	0.097	0.126	-0.05	0.003	-0.088	-0.188	-0.017	0.04	-0.108
PExpCRVD	0.108	**0.941**	-0.018	-0.009	-0.034	-0.092	-0.006	-0.015	-0.001	-0.151
PExcCRV	0.185	-0.015	**0.939**	-0.01	-0.045	-0.019	-0.044	0.04	0.036	-0.017
PExcCRVD	0.086	-0.017	**0.949**	-0.003	-0.015	0.002	0.037	0.014	0.008	0.02
PTCRV	0.326	-0.039	-0.03	0.009	-0.04	**0.847**	-0.021	0.048	-0.04	-0.014
PTCRVD	-0.131	0.052	0.009	-0.01	-0.017	**0.909**	0.064	-0.002	0.013	0.048
PConCRV	0.032	-0.005	-0.008	**0.962**	-0.015	0.009	-0.011	0.003	0.021	-0.004
PConCRVD	0.008	-0.004	-0.004	**0.961**	-0.01	-0.01	-0.005	-0.017	0.039	0.004
PComCRV	**0.647**	-0.11	0.054	-0.028	0.112	0.059	0.537	0.183	-0.204	-0.097
PComCRVD	-0.117	0.198	0.017	-0.006	0.054	0.075	**0.775**	0.141	-0.031	-0.185
PSCV	**0.907**	-0.055	0.126	0.037	0.211	0.145	0.041	0.169	0.005	0.078
PSCVD	-0.158	**0.946**	-0.011	0	0.012	0.066	0.11	0.019	0.069	0.173

Table C.5 PCA for Coding standard violations-based metrics (Camel)

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10
PSRV	-0.105	**0.68**	0.416	0.046	0.417	0.193	0.205	0.228	0.087	0.121
PSRVD	**0.975**	0.006	0.018	0.007	0.077	-0.013	0.033	0.029	0.007	0
PNCRV	0.02	**0.939**	0.099	0.02	0.136	0.026	0.028	0.042	0.019	0.05
PNCRVD	**0.673**	0.569	-0.171	-0.004	-0.116	-0.06	-0.149	-0.112	-0.02	-0.081
PPCICRV	-0.091	0.196	0.062	0.067	0.09	0.069	0.024	**0.88**	0.016	-0.023
PPCICRVD	0.101	-0.049	0.011	0.014	-0.029	0.005	-0.033	**0.901**	-0.013	0.122
PFCRV	-0.171	0.32	0.153	0.005	**0.853**	0.049	-0.028	0.102	0.037	0.051
PFCRVD	0.22	0.029	-0.043	0.004	**0.902**	-0.029	-0.095	-0.044	-0.039	0.033
PMCRV	-0.059	0.097	0.125	-0.011	0.038	**0.925**	0.007	0.051	0.069	0.008
PMCRVD	0.015	0.019	0.007	-0.001	-0.005	**0.94**	-0.004	0.02	-0.02	0.006
PDSCRV	-0.071	0.219	0.073	-0.02	-0.021	0.02	**0.911**	0.028	-0.009	0.034
PDSCRVD	0.092	-0.091	-0.057	0.002	-0.068	-0.018	**0.916**	-0.038	-0.019	-0.041
PExpCRV	-0.158	0.27	**0.861**	0.058	0.113	0.153	0.034	0.061	0.05	0.011
PExpCRVD	0.103	0.001	**0.93**	0.026	-0.019	-0.012	-0.04	-0.002	-0.036	0.002
PExcCRV	0.002	0.021	0.039	**0.998**	0.007	-0.006	-0.008	0.04	-0.002	0.003
PExcCRVD	0.002	0.021	0.039	**0.998**	0.007	-0.006	-0.008	0.04	-0.002	0.003
PTCRV	-0.046	0.07	0.042	-0.006	0.034	0.071	-0.004	0.015	**0.894**	-0.006
PTCRVD	0.06	-0.009	-0.026	0.002	-0.03	-0.026	-0.02	-0.011	**0.895**	-0.016
PComCRV	-0.286	0.362	0.143	0.013	0.188	0.07	0.057	0.082	0.008	**0.79**
PComCRVD	0.14	-0.079	-0.057	-0.002	-0.025	-0.028	-0.043	0.053	-0.029	**0.946**
PSCV	-0.05	**0.616**	0.433	0.043	0.391	0.212	0.268	0.226	0.103	0.176
PSCVD	**0.96**	-0.118	0.014	0	0.03	-0.011	0.049	0.03	0.019	0.009

Table C.6: PCA for Coding standard violations-based metrics (Ant)

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11
PSRV	**.905**	-.160	.016	.174	.175	.079	.151	.169	.029	.139	.096
PSRVD	-.099	**.925**	-.008	.023	.007	.000	.073	.168	.140	.185	.160
PNCRV	**.874**	.163	.027	-.010	.014	-.009	-.041	-.103	-.052	-.137	-.114
PNCRVD	.000	**.906**	-.009	-.084	-.056	-.021	-.118	-.084	-.160	-.125	-.141
PPCICRV	.293	-.097	-.017	**.858**	.075	.018	.033	.018	-.034	.013	-.063
PPCICRVD	-.012	.058	.000	**.929**	-.004	-.018	-.028	-.026	.013	.003	.007
PFCRV	**.732**	-.227	-.005	-.011	.056	.048	.080	.051	-.019	.003	.530
PFCRVD	.099	.150	-.005	-.046	-.029	-.009	.034	-.061	.059	-.004	**.952**
PMCRV	.304	-.102	-.016	.066	**.858**	.027	-.001	.044	-.048	.015	-.028
PMCRVD	.023	.051	.001	.004	**.924**	-.016	-.031	-.035	.040	.067	.007
PDSCRV	.397	-.079	.039	.016	.026	.104	.006	**.793**	-.004	.053	-.017
PDSCRVD	-.085	.303	-.009	-.027	-.020	-.022	-.022	**.857**	.005	-.047	-.041
PExpCRV	**.681**	-.209	-.006	.103	.165	-.010	.045	.101	-.081	.549	-.027
PExpCRVD	.075	.215	-.003	-.011	.047	-.011	.048	-.027	.010	**.937**	.002
PExcCRV	.129	-.041	-.007	.017	.028	**.903**	.023	.073	-.012	-.026	.006
PExcCRVD	-.015	.030	.000	-.018	-.020	**.916**	-.028	-.013	.003	.014	-.004
PTCRV	.504	-.147	-.031	.042	-.006	.021	**.758**	.027	-.040	-.034	-.031
PTCRVD	-.017	.103	.001	-.025	-.029	-.020	**.936**	-.030	.021	.078	.068
PConCRV	.021	-.010	**.999**	-.007	-.006	-.003	-.009	.011	-.002	-.002	-.003
PConCRVD	.021	-.010	**.999**	-.007	-.006	-.003	-.009	.011	-.002	-.002	-.003
PComCRV	.561	-.215	.013	.042	.039	.010	.042	.014	**.725**	-.025	.073
PComCRVD	-.148	.195	-.010	-.039	-.020	-.016	-.023	-.005	**.926**	.006	.022
PSCV	**.848**	-.114	.033	.216	.190	.114	.177	.205	.124	.150	.131
PSCVD	-.191	**.903**	-.007	.030	.000	.008	.074	.179	.151	.174	.140

Table C.7: PCA for Coding standard violations-based metrics (All systems)

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11
PSRV	**.927**	-.076	.021	.090	.172	.159	.131	.153	.028	.131	.028
PSRVD	-.162	**.892**	-.003	-.002	.061	.036	.052	.111	.076	.125	.309
PNCRV	**.649**	-.100	.007	-.024	.035	-.041	-.030	.078	.031	-.094	.599
PNCRVD	-.174	.327	-.008	-.010	-.047	-.014	-.015	-.010	-.055	-.018	**.855**
PPCICRV	.366	-.061	-.015	-.026	**.830**	.037	-.018	-.054	-.062	.046	-.025
PPCICRVD	-.025	.087	-.002	-.004	**.921**	.030	-.017	-.007	.026	.001	-.002
PFCRV	.603	-.125	.003	.084	-.044	.052	.085	**.682**	.019	-.050	-.059
PFCRVD	.036	.123	-.007	-.007	-.031	.009	.003	**.945**	.069	-.009	.048
PMCRV	.359	-.079	.038	.080	.011	**.820**	.039	-.031	-.034	.004	-.037
PMCRVD	.030	.106	.020	-.011	.052	**.905**	-.022	.055	.044	.055	.004
PDSCRV	.566	-.095	.041	.043	.085	.064	.042	-.067	-.029	**.680**	-.093
PDSCRVD	-.016	.150	.001	-.009	-.003	.029	-.037	.001	.015	**.919**	.005
PExpCRV	**.839**	.093	-.020	.048	.002	.136	.001	-.085	-.072	-.033	-.203
PExpCRVD	.274	**.843**	-.007	-.026	-.058	.015	-.079	-.084	-.038	-.083	-.256
PExcCRV	.142	-.020	-.006	**.914**	-.022	.067	.016	.021	-.010	.003	-.023
PExcCRVD	.022	.004	.000	**.927**	-.001	-.013	-.009	.008	.008	.009	.006
PTCRV	.347	-.064	.005	.016	-.038	.023	**.824**	.031	-.005	.003	-.039
PTCRVD	-.069	.079	-.003	-.007	.001	-.010	**.905**	.013	.013	-.019	.014
PConCRV	.020	-.006	**.966**	-.003	-.009	.019	.005	.000	-.003	.007	-.003
PConCRVD	.003	-.003	**.965**	-.002	-.005	.032	-.004	-.005	-.007	.013	.000
PComCRV	.540	-.164	-.009	.016	-.052	.026	.029	.105	**.720**	-.016	-.084
PComCRVD	-.143	.168	-.006	-.010	.006	.004	-.005	.025	**.922**	.011	.013
PSCV	**.882**	-.047	.029	.103	.206	.146	.147	.158	.085	.158	.011
PSCVD	-.217	**.887**	-.002	.004	.055	.010	.065	.070	.074	.111	.268

Table C.8 PCA for Coding standard violations-based AND CK metrics (Velocity)

	PC1	PC 2	PC 3	PC 4	PC 5	PC 6	PC 7	PC 8	PC 9	PC 10
PSRV	**0.835**	-0.07	0.337	0.16	0.089	0.096	0.218	0.211	0.06	0.076
PSRVD	-0.131	**0.937**	-0.105	0.009	0.016	-0.044	0.012	0.188	0.031	0.002
PNCRV	**0.609**	0.156	0.106	-0.088	0.053	-0.091	0.168	-0.199	-0.032	0.311
PNCRVD	-0.135	**0.826**	-0.031	-0.072	-0.01	-0.105	0.003	-0.132	-0.097	0.111
PPCICRV	0.26	-0.06	0.413	0.085	0.013	0.013	**0.821**	0.014	-0.024	-0.055
PPCICRVD	0.174	-0.014	-0.068	0.015	-0.027	0.062	**0.938**	-0.013	0.029	0.002
PFCRV	**0.774**	-0.05	0.083	0.14	0.066	0.02	0.126	0.184	-0.314	-0.085
PFCRVD	0.388	**0.649**	-0.127	0.042	0.004	-0.039	-0.026	0.091	-0.326	-0.094
PMCRV	0.186	-0.116	0.146	0.039	0.039	**0.864**	0.027	0.062	-0.119	0.000
PMCRVD	0.013	0.013	-0.022	0.011	-0.015	**0.921**	0.043	0.028	0.01	0.002
PDSCRV	0.456	-0.062	0.356	0.049	0.001	0.106	0.021	**0.689**	-0.01	0.066
PDSCRVD	0.015	0.144	-0.096	-0.049	0.005	0.034	-0.015	**0.839**	0.01	-0.008
PExpCRV	**0.695**	-0.188	0.272	0.109	0	-0.095	0.061	0.145	0.413	0.018
PExpCRVD	0.444	0.053	-0.123	0.109	-0.005	-0.126	-0.029	0.068	**0.723**	-0.027
PExcCRV	0.094	-0.017	0.095	0.036	**0.941**	0.04	0.009	-0.024	-0.045	0.03
PExcCRVD	0.028	0.002	-0.044	-0.029	**0.943**	-0.021	-0.022	0.034	-0.037	0.005
PTCRV	0.146	-0.048	0.216	**0.912**	0.042	0.041	0.09	0.007	-0.077	-0.041
PTCRVD	0.041	-0.033	-0.102	**0.943**	-0.033	0.008	-0.006	-0.027	-0.071	0.003
PSCV	**0.846**	-0.048	0.153	0.184	0.138	0.169	0.231	0.207	0.067	0.039
PComCRV	**0.773**	-0.034	0.087	-0.121	-0.085	0.238	-0.061	-0.2	0.223	-0.005
PComCRVD	0.308	**0.661**	-0.051	-0.105	-0.092	0.14	-0.165	-0.259	0.144	-0.052
PSCVD	-0.242	**0.886**	-0.106	0.011	0.032	-0.037	0.027	0.221	0.031	-0.034
wmc	0.277	-0.159	**0.885**	0.018	0.021	0.066	0.075	0.011	-0.116	0.057
dit	-0.066	-0.061	-0.045	-0.164	-0.056	-0.007	0.022	-0.028	**0.563**	-0.014
noc	0.087	0.019	0.005	0.007	0.028	-0.037	-0.046	0.067	-0.047	**0.913**
cbo	0.031	-0.084	**0.569**	-0.09	0.004	0.156	0.016	-0.105	0.081	0.523
rfc	0.522	-0.241	**0.732**	0.022	0.055	0.091	0.114	-0.025	-0.079	0.014
lcom	0.071	-0.009	**0.918**	0.093	0.01	-0.032	0.049	0.069	-0.025	-0.033

Table C.9 PCA for Coding standard violations-based with CK metrics (Poi)

	PC1	PC 2	PC 3	PC 4	PC 5	PC 6	PC 7	PC 8	PC 9	PC10	PC11
PSRV	.413	-.138	**.536**	.406	.198	.389	.363	.043	.046	.017	.058
PSRVD	-.163	**.939**	.013	.132	-.037	.099	.104	.000	.055	.100	-.066
PNCRV	.306	.159	**.584**	.281	.066	.179	.172	-.011	-.085	-.168	.069
PNCRVD	-.024	**.868**	.110	-.070	-.035	-.143	-.076	-.021	-.119	-.082	.027
PPCICRV	.163	-.075	.126	**.908**	.034	.083	.125	-.050	-.067	-.040	.017
PPCICRVD	-.125	.235	-.191	**.838**	-.047	.002	-.047	-.011	.095	-.055	-.036
PFCRV	.208	-.160	**.844**	-.023	.148	.027	-.015	.091	.147	.050	-.001
PFCRVD	-.126	.195	**.793**	-.159	-.021	-.010	-.129	.003	.147	.003	-.060
PMCRV	.117	-.073	.111	.010	**.939**	.002	.031	.066	.017	.034	-.014
PMCRVD	-.018	-.004	.052	.004	**.951**	.021	.000	-.002	.056	.001	-.008
PDSCRV	.322	-.172	.173	.146	.080	.194	**.802**	.009	-.025	.015	.080
PDSCRVD	-.097	.271	-.123	-.014	-.048	.071	**.882**	-.002	-.018	-.066	-.049
PExpCRV	.332	-.221	.218	.068	.066	**.800**	.173	.020	-.006	.005	.033
PExpCRVD	-.088	.179	-.056	.029	-.050	**.914**	.064	.026	.047	-.054	-.054
PExcCRV	-.007	-.016	.065	-.023	.070	-.008	.005	**.963**	.008	-.005	.000
PExcCRVD	-.011	-.002	.022	-.024	-.007	.050	.006	**.963**	.021	.001	.003
PTCRV	.293	.024	.040	-.024	.046	.018	.004	-.019	.031	**.821**	.060
PTCRVD	-.048	.139	-.078	-.042	-.009	-.040	-.090	-.003	-.039	**.859**	-.055
PComCRV	.203	-.212	.302	-.023	.120	.060	.058	.037	**.826**	.003	.105
PComCRVD	-.089	.163	.017	.043	-.009	.010	-.066	.003	**.943**	-.009	-.019
PSCV	.316	-.119	**.467**	.390	.242	.440	.384	.041	.112	.009	.053
PSCVD	-.199	**.913**	-.099	.076	-.026	.059	.082	.001	.088	.133	-.068
wmc	**.886**	-.222	.119	.110	.061	.076	.051	-.033	-.047	.033	.034
dit	.155	.209	-.345	.072	-.008	.074	-.201	-.059	-.017	**-.364**	-.148
noc	-.095	.011	-.037	.013	.007	-.035	.005	-.005	.031	.020	**.867**
cbo	.279	-.096	.058	-.018	-.030	.021	.015	.008	.021	.013	**.774**
rfc	**.855**	-.216	.225	.087	.090	.072	.103	.008	.046	.117	.124
lcom	**.855**	.014	-.072	-.083	-.020	.021	.002	.000	.058	.045	.021

Table C.10 PCA for Coding standard violations-based with CK metrics (Xalan)

	PC1	PC 2	PC 3	PC 4	PC 5	PC 6	PC 7	PC 8	PC 9	PC10	PC11
PSRV	**.881**	.343	-.057	.154	.034	.167	.083	-.033	.139	.035	-.011
PSRVD	-.089	-.086	**.966**	-.015	-.003	.040	.065	.022	.075	.150	.012
PNCRV	**.624**	-.124	.044	-.071	.009	.068	.103	-.456	.056	-.095	.227
PNCRVD	-.207	-.251	.441	-.087	-.006	-.004	.104	**-.497**	.009	-.039	.240
PPCICRV	.386	.069	-.090	-.081	-.037	**.756**	-.054	.042	-.141	-.119	.038
PPCICRVD	.016	-.068	.044	-.024	-.022	**.874**	-.005	.033	-.016	.029	.066
PFCRV	.563	.245	-.095	.163	.001	-.074	.040	.003	**.640**	-.114	.010
PFCRVD	.097	-.110	.086	-.017	-.023	-.041	.023	.092	**.911**	.082	.036
PMCRV	.353	**.491**	-.018	.220	.128	.324	-.029	-.159	.138	.135	-.225
PMCRVD	-.033	.166	.186	.091	.093	**.502**	-.008	-.081	.380	.464	-.146
PDSCRV	.751	.299	-.124	-.017	.049	-.010	-.019	-.006	-.074	.351	.032
PDSCRVD	.121	-.095	.156	-.052	.004	-.024	-.035	.091	.016	**.868**	.062
PExpCRV	**.702**	.332	.066	.179	-.029	.057	-.143	-.026	-.004	-.052	-.136
PExpCRVD	.067	.076	**.889**	.022	-.007	-.027	-.136	.155	-.025	-.004	-.127
PExcCRV	.171	.004	-.015	**.937**	-.007	-.029	-.016	-.058	.043	-.024	-.033
PExcCRVD	.098	-.054	-.008	**.932**	-.007	-.021	.018	-.023	.009	-.007	.034
PTCRV	.273	.256	-.042	-.003	.014	-.042	**.815**	.048	.044	-.054	.071
PTCRVD	-.089	-.074	.057	.004	-.010	-.012	**.918**	.032	.002	.008	-.042
PConCRV	.030	.004	-.007	-.010	**.961**	-.014	.009	.010	.000	-.003	-.007
PConCRVD	.005	-.005	-.005	-.004	**.959**	-.005	-.008	-.011	-.012	.024	.002
PComCRV	**.694**	.041	-.131	.053	-.053	-.009	.061	.484	.105	.102	.104
PComCRVD	-.062	-.140	.192	.006	-.031	-.029	.100	**.699**	.102	.279	.092
PSCV	**.899**	.192	-.057	.141	.040	.203	.127	.030	.146	.005	.017
PSCVD	-.117	-.100	**.964**	-.021	-.004	.017	.076	.028	.025	.078	.019
wmc	.358	**.835**	-.080	-.050	-.008	-.001	.051	-.089	-.036	-.016	.125
dit	-.011	-.153	.122	-.169	.035	.081	.040	**.530**	.004	-.197	.074
noc	.009	.114	-.020	.026	.006	.035	-.006	.011	-.028	.092	**.818**
cbo	.143	.481	-.062	-.048	-.021	.038	.032	.118	.143	-.150	**.557**
rfc	.529	**.728**	-.101	.000	.003	-.019	.071	.002	.053	-.074	.102
lcom	.112	**.822**	-.011	-.045	-.017	-.023	.050	-.076	-.062	-.001	.113

Table C.11 PCA for Coding standard violations-based with CK metrics (Camel)

	PC1	PC 2	PC 3	PC 4	PC 5	PC 6	PC 7	PC 8	PC 9	PC10	PC11	PC12
PSRV	-.015	-.102	**.701**	.401	.047	.405	.193	.201	.225	.085	.113	.006
PSRVD	-.024	**.973**	-.004	.019	.008	.081	-.011	.035	.031	.006	.001	.001
PNCRV	-.006	.030	**.937**	.089	.018	.125	.021	.022	.039	.018	.044	.003
PNCRVD	.013	**.683**	.544	-.166	-.008	-.122	-.063	-.151	-.111	-.018	-.079	.077
PPCICRV	.005	-.089	.204	.061	.066	.085	.067	.022	**.878**	.017	-.023	.003
PPCICRVD	-.016	.100	-.040	.009	.016	-.031	.006	-.033	**.899**	-.013	.123	-.033
PFCRV	.016	-.170	.345	.144	.005	**.842**	.050	-.032	.098	.037	.049	.029
PFCRVD	.015	.220	.043	-.041	.004	**.899**	-.030	-.096	-.046	-.038	.036	-.024
PMCRV	.000	-.059	.105	.121	-.011	.035	**.924**	.006	.050	.069	.007	-.022
PMCRVD	-.002	.015	.020	.009	-.002	-.006	**.939**	-.004	.020	-.020	.007	-.020
PDSCRV	-.009	-.070	.226	.069	-.021	-.024	.019	**.909**	.027	-.009	.031	.001
PDSCRVD	-.015	.091	-.085	-.056	.003	-.070	-.018	**.915**	-.039	-.019	-.039	.007
PExpCRV	-.001	-.162	.296	**.845**	.060	.105	.157	.033	.059	.048	.005	.001
PExpCRVD	-.006	.097	.019	**.924**	.028	-.020	-.009	-.038	-.002	-.036	.000	-.044
PExcCRV	.009	.003	.022	.040	**.998**	.006	-.006	-.008	.040	-.002	.004	.014
PExcCRVD	.009	.003	.022	.040	**.998**	.006	-.006	-.008	.040	-.002	.004	.014
PTCRV	-.018	-.047	.075	.038	-.006	.033	.072	-.004	.015	**.894**	-.008	-.003
PTCRVD	-.014	.060	-.009	-.025	.002	-.030	-.027	-.019	-.011	**.895**	-.015	-.022
PComCRV	-.025	-.287	.375	.135	.012	.184	.070	.055	.082	.006	**.783**	.021
PComCRVD	-.030	.134	-.075	-.058	-.001	-.022	-.026	-.041	.054	-.030	**.945**	.004
PSCV	-.027	-.048	**.635**	.419	.043	.381	.211	.265	.224	.101	.168	.008
PSCVD	-.033	**.956**	-.126	.015	.001	.036	-.008	.052	.032	.018	.009	.006
wmc	**.954**	-.020	.013	-.007	.003	.015	.001	-.015	-.006	-.026	-.030	.067
dit	.133	.045	.011	.120	-.028	-.038	-.002	-.034	-.023	.042	.061	**-.809**
noc	.224	.073	.084	-.008	.014	-.083	-.018	-.032	-.072	.006	.049	**.435**
cbo	.379	.007	-.108	.208	-.033	.094	-.046	.016	.044	.018	.052	**.476**
rfc	**.910**	.003	.009	.017	.029	.029	.010	.005	-.014	-.018	-.005	.029
lcom	**.857**	-.041	-.023	-.050	-.007	-.032	-.001	-.017	.003	.004	-.028	.055

Table C.12 PCA for Coding standard violations-based with CK metrics (Ant)

	PC1	PC 2	PC 3	PC 4	PC 5	PC 6	PC 7	PC 8	PC 9	PC10	PC11	PC12
PSRV	**.844**	-.149	.261	.019	.195	.179	.077	.201	.159	.039	.185	-.029
PSRVD	-.037	**.937**	-.172	-.011	.005	.003	.001	.124	.073	.139	.165	.021
PNCRV	**.701**	.102	.233	.039	.091	.060	-.012	.042	-.026	-.073	-.062	.005
PNCRVD	-.102	**.866**	.022	.001	-.037	-.027	-.017	-.016	-.111	-.188	-.100	-.050
PPCICRV	.236	-.097	.137	-.013	**.852**	.081	.024	.028	.044	-.034	.040	.006
PPCICRVD	.003	.058	-.053	-.001	**.910**	-.003	-.013	-.047	-.021	.017	.004	.055
PFCRV	**.901**	-.161	.054	-.013	-.061	.023	.043	-.040	.077	.011	.000	.015
PFCRVD	**.539**	.262	-.388	-.033	-.171	-.103	-.022	-.297	.016	.114	-.090	.149
PMCRV	.259	-.081	.222	-.015	.060	**.848**	.028	.041	-.003	-.037	.029	-.028
PMCRVD	.042	.040	-.063	-.004	.013	**.923**	-.021	-.032	-.034	.039	.059	.033
PDSCRV	.373	-.069	.067	.038	.011	.027	.105	**.794**	.017	-.001	.080	.075
PDSCRVD	-.067	.322	-.062	-.013	-.044	-.030	-.022	**.827**	-.024	.020	-.065	.006
PExpCRV	**.581**	-.192	.289	-.003	.124	.162	-.012	.132	.046	-.066	.580	-.095
PExpCRVD	.054	.217	-.057	-.009	.008	.040	-.019	-.024	.041	.025	**.919**	-.008
PExcCRV	.127	-.033	.061	-.008	.021	.023	**.900**	.073	.018	-.005	-.029	-.032
PExcCRVD	-.010	.026	-.027	-.001	-.015	-.019	**.914**	-.010	-.028	-.003	.013	.020
PTCRV	.425	-.144	.231	-.026	.066	-.002	.019	.064	**.759**	-.034	-.008	-.086
PTCRVD	.015	.106	-.064	-.004	-.027	-.036	-.025	-.046	**.931**	.024	.062	.029
PConCRV	.023	-.008	.001	**.999**	-.008	-.008	-.004	.010	-.010	-.002	-.003	.004
PConCRVD	.023	-.008	.001	**.999**	-.008	-.008	-.004	.010	-.010	-.002	-.003	.004
PComCRV	.520	-.226	.099	.021	.056	.060	.016	.049	.057	**.713**	.017	.005
PComCRVD	-.148	.190	-.100	-.002	-.068	-.002	-.001	-.021	-.005	**.896**	.023	.017
PSCV	**.814**	-.101	.197	.035	.230	.192	.113	.224	.184	.134	.187	-.011
PSCVD	-.126	**.921**	-.159	-.010	.005	-.009	.010	.126	.072	.154	.146	.005
wmc	.471	-.230	**.773**	-.008	.099	.055	.012	.039	.052	-.027	-.027	-.024
dit	.078	.015	.119	-.064	.229	-.110	-.085	.044	-.123	.328	-.209	**-.410**
noc	.018	-.014	.103	-.019	.115	-.024	-.036	.076	-.059	.056	-.088	**.836**
cbo	.038	-.005	.651	.015	.014	-.005	-.011	-.028	-.006	.015	.014	**.482**
rfc	.593	-.282	**.648**	.034	.097	.104	.076	.079	.083	.011	.008	-.076
lcom	.187	-.038	**.860**	-.017	-.024	.052	.012	-.035	.024	-.010	.000	.003

Appendix D: Multivariate Regression Models

The tables, Table D.1 through Table D.6, present the multivariate logistic and linear regression models built to predict the fault-proneness and the fault density, respectively for the six systems under study. At each system from the target set, the models are built in three different ways. (1) Using subsets of both CK and coding standard violations-based metrics, (2) using a subset of only coding standard violations-based metrics, and (3) using a subset of only CK metrics.

The tables can be read as follows. The regression coefficient B*i,j* of each metric at each system appears inside corresponding cell in the table of coefficients. The coefficient *Bi,j* for the metric in column *j* at system *i* is located in the intersection of the column *j* with the row *i*. For example, referring to Table D.2 the coefficient *B2,6* is located at the intersection of column 6 (PPCICRVD) with row 2 (Velocity), which is 4.7.

Each row in the table represents a system. The first row corresponding to Synapse in the table, the second row corresponding to Velocity, the third row corresponding to Poi, the fourth row corresponding to Xalan, the fifth row corresponding to Camel and the sixth row corresponding to Ant. Each column in the table represents a metric, and indexed from j=1 for PSRV metric till j=25 for the intercept.

D.1 Regression Models

Tables D.1 through D.3 represent the multivariate linear regression models for predicting the fault density for all systems under study. While Tables D.4 through D.6 represent the multivariate logistic regression models for predicting the fault-proneness for all systems under study.

$\mathbf{M_i}$ denotes the Regression Model at system i . For example, referring to the linear regression models in Table D.2,

$$M_5 = B_{5,1} \times PSRV + B_{5,2} \times PSRVD + B_{5,3} \times PNCRV + B_{5,4} \times PNCRVD + B_{5,5} \times PPCICRV + B_{5,6} \times PPCICRVD + B_{5,7} \times PFCRV + B_{5,8} \times PFCRVD + B_{5,9} \times PMCRV + B_{5,10} \times PMCRVD + B_{5,11} \times PDSCRV + B_{5,12} \times PDSCRVD + B_{5,13} \times PExpCRV + B_{5,14} \times PExpCRVD + B_{5,15} \times PExcCRV + B_{5,16} \times PExcCRVD + B_{5,17} \times PTCRV + B_{5,18} \times PTCRVD + B_{5,19} \times PConCRV + B_{5,20} \times PConCRVD + B_{5,21} \times PComCRV + B_{5,22} \times PComCRVD + B_{5,23} \times PSCV + B_{5,24} \times PSCVD + B_{5,25}$$

$$= -7.07PSRVD + 0.2PNCRVD + 8PPCICRVD - 0.04PMCRV + 0.6PDSCRVD + 2.2PTCRVD - 0.06PConCRV - 0.05PComCRV + 3.70PSCVD + 9.3$$

Table D.1: Multivariate linear regression models built based on CK metrics, metrics' coefficients (all systems)

	WMC	DIT	NOC	CBO	RFC	LCOM	Intercept
	1	2	3	4	5	6	7
Synapse	0	-0.8391	0	0	0	0	10.9136
Velocity	0	-0.6667	0	0	0	0	10.3058
Poi	0	-0.521	0.0936	0	-0.0358	0	10.5064
Xalan	0	-0.7222	0	0.0463	0	0	10.317
Camel	0	-1.2219	0	0	-0.0276	0	11.627
Ant	-0.0724	-0.6746	0.1765	0	0	0	12.7291
The model at release i is given by the following equation. $M_i = B_{i,1} \times WMC + B_{i,2} \times DIT + B_{i,3} \times NOC + B_{i,4} \times CBO + B_{i,5} \times RFC + B_{i,6} \times LCOM + B_{i,7}$							

Table D.2: Multivariate linear regression models built based on coding standard violations-based metrics, metrics' coefficients (all systems)

system	PSRV	PSRVD	PNCRV	PNCRVD	PPCICRV	PPCICRVD	PFCRV	PFCRVD	PMCRV	PMCRVD	PDSCRV	PDSCRVD	PExpCRV	PExpCRVD	PExcCRV	PExcCRVD	PTCRV	PTCRVD	PConCRV	PConCRVD	PComCRV	PComCRVD	PSCV	PSCVD	intercept
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Synapse	0	0	0	0	0	0	0	0	0	-11.3	0	0	0	0	0	0	0	0	0	0	0	0	0	5.17	6.1
Velocity	0	-24.2	0	0	0	4.7	0	3.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10.7	5.4
Poi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4.94	5.39
Xalan	0	0	0	0	0	0	0	0	0	-12.4	0	0	0	0	0	3.3	0	0	0	-42.8	0	0	0	7.75	4.7
Camel	0	-7.07	0	0.2	0	8	0	0	-0.04	0	0	0.6	0	0	0	0	0	2.2	-0.06	0	-0.05	0	0	3.70	9.3
Ant	0	0	0	0	0	0	0	3.76	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5.42	6.73

The model at system i is given by the following:

$$M_i = B_{i,1} \times PSRV + B_{i,2} \times PSRVD + B_{i,3} \times PNCRV + B_{i,4} \times PNCRVD + B_{i,5} \times PPCICRV + B_{i,6} \times PPCICRVD + B_{i,7} \times PFCRV + B_{i,8} \times PFCRVD + B_{i,9} \times PMCRV + B_{i,10} \times PMCRVD + B_{i,11} \times PDSCRV + B_{i,12} \times PDSCRVD + B_{i,13} \times PExpCRV + B_{i,14} \times PExpCRVD + B_{i,15} \times PExcCRV + B_{i,16} \times PExcCRVD + B_{i,17} \times PTCRV + B_{i,18} \times PTCRVD + B_{i,19} \times PConCRV + B_{i,20} \times PConCRVD + B_{i,21} \times PComCRV + B_{i,22} \times PComCRVD + B_{i,23} \times PSCV + B_{i,24} \times PSCVD + B_{i,25}$$

Table D.3: Multivariate linear regression models built based on coding standard violations-based metrics and CK metrics, metrics' coefficients (all systems)

	PSRV	PSRVD	PNCRV	PNCRVD	PPCICRV	PPCICRVD	PFCRV	PFCRVD	PMCRV	PMCRVD	PDSCRV	PDSCRVD	PExpCRV	PExpCRVD	PExcCRV
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Synapse	0	0	0	0	0	0	0	0	0	-11.21	0	0	0	0	0
Velocity	0	-25.40	0	0	0	4.59	0	3.63	0	0	0	0	0	0	0
Poi	0	0	0	0	0	3.07	0	0	0	0	0	0	0	0	0
Xalan	0	0	0	0	0	0	0	0	0	-12.36	0	0	0	0	0
Camel	0	-9.71	0	0	0	7.21	0	0	0	0	0	0	0	0	0
Ant	0	0	0	0	0	0	0	3.75	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	PExcCRVD	PTCRV	PTCRVD	PConCRV	PConCRVD	PComCRV	PComCRVD	PSCV	PSCVD	WMC	DIT	NOC	CBO	RFC	LCOM	Intercept
Synapse	0	0	0	0	0	0	0	0	5.22	0	-0.91	0	0	0	0	8.11
Velocity	0	0	0	0	0	0	0	0	11.01	0	-0.74	0	0	0	0	6.99
Poi	0	0	0	0	0	0	0	0	5.18	0	-0.83	0	0	0	0	7.63
Xalan	3.3	0	0	0	-42.80	0	0	0	7.75	0	0	0	0	0	0	4.6785
Camel	0	0	0	0	0	-0.05	0	0	4.92	0	-1.10	0	0	0	0	11.73
Ant	0	0	0	0	0	0	0	0	5.43	0	0	0.17	0	0	0	6.63

$$M_i = B_{i,1} \times PSRV + B_{i,2} \times PSRVD + B_{i,3} \times PNCRV + B_{i,4} \times PNCRVD + B_{i,5} \times PPCICRV + B_{i,6} \times PPCICRVD + B_{i,7} \times PFCRV + B_{i,8} \times PFCRVD + B_{i,9} \times PMCRV + B_{i,10} \times PMCRVD + B_{i,11} \times PDSCRV + B_{i,12} \times PDSCRVD + B_{i,13} \times PExpCRV + B_{i,14} \times PExpCRVD + B_{i,15} \times PExcCRV + B_{i,16} \times PExcCRVD + B_{i,17} \times PTCRV + B_{i,18} \times PTCRVD + B_{i,19} \times PConCRV + B_{i,20} \times PConCRVD + B_{i,21} \times PComCRV + B_{i,22} \times PComCRVD + B_{i,23} \times PSCV + B_{i,24} \times PSCVD + B_{i,25} \times WMC + B_{i,26} \times DIT + B_{i,27} \times NOC + B_{i,28} \times CBO + B_{i,29} \times RFC + B_{i,30} \times LCOM + B_{i,31}$$

Table D.4: Multivariate logistic regression models built based on coding standard violations-based metrics, metrics' coefficients (all systems)

system	PSRV	PSRVD	PNCRV	PNCRVD	PPCICRV	PPCICRVD	PFCRV	PFCRVD	PMCRV	PMCRVD	PDSCRV	PDSCRVD	PExpCRV	PExpCRVD	PExcCRV	PExcCRVD	PTCRV	PTCRVD	PConCRV	PConCRVD	PComCRV	PComCRVD	PSCV	PSCVD	intercept
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Synapse	0	8.78	-0.01	0.01	-0.05	0.90	0	0	0	0.82	-0.03	0.8	-0.01	-0.07	0	0	0	0	0	0	0	0	-0.02	-1.76	1.7573
Velocity	0	4.78	0	-0.03	-0.05	0.96	0	0	0	1.18	-0.02	0.77	-0.01	-0.08	0	0	0	0	0	0	0	0	-0.03	-0.74	1.7268
Poi	0	- 6.11	0	0.007	0	-0.23	0	0	0	- 0.50	0	-0.63	0	-0.51	0	0	0	-0.42	0	0	0	-0.1	0.04	1.04	-1.7817
Xalan	0	0	0	0.03	-0.05	1.08	0	0.2	0	2.22	-0.02	0.65	-0.023	0.62	0	0	0	0	0	0	0	0	-0.04	0.04	2.1582
Camel	0	0	0	-0.08	-0.04	0	0	0	0	0	0	-0.14	0.002	0	0	0	0	0	0	0	0	0	-0.04	1.03	1.4294
Ant	0	- 7.8	0.004	0.52	0.016	-0.56	0	0	0	- 0.01	-0.002	-0.48	0.01	0.99	0	0	0	0	0	0	0	0	0.06	- 1.29	-2.8102

The model at system *i* is given by the following equation.

$$P_i = B_{i,1} \times PSRV + B_{i,2} \times PSRVD + B_{i,3} \times PNCRV + B_{i,4} \times PNCRVD + B_{i,5} \times PPCICRV + B_{i,6} \times PPCICRVD + B_{i,7} \times PFCRV + B_{i,8} \times PFCRVD + B_{i,9} \times PMCRV + B_{i,10} \times PMCRVD + B_{i,11} \times PDSCRV + B_{i,12} \times PDSCRVD + B_{i,13} \times PExpCRV + B_{i,14} \times PExpCRVD + B_{i,15} \times PExcCRV + B_{i,16} \times PExcCRVD + B_{i,17} \times PTCRV + B_{i,18} \times PTCRVD + B_{i,19} \times PConCRV + B_{i,20} \times PConCRVD + B_{i,21} \times PComCRV + B_{i,22} \times PComCRVD + B_{i,23} \times PSCV + B_{i,24} \times PSCVD + B_{i,25}$$

$$Then\ M_i = \frac{e^{P_i}}{1 + e^{P_i}}$$

Table D.5: Multivariate logistic regression models built based on coding standard violations-based metrics and CK metrics, metrics' coefficients (all systems)

	PSRV	PSRVD	PNCRV	PNCRVD	PPCICRV	PPCICRVD	PFCRV	PFCRVD	PMCRV	PMCRVD	PDSCRV	PDSCRVD	PExpCRV	PExpCRVD	PExcCRV
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Synapse	0	8.63	- 0.01	0.010	- 0.06	0.92	0	0	0	0.80	- 0.03	0.73	- 0.01	- 0.16	0
Velocity	0	7.73	- 0.01	0.04	- 0.06	1.02	0	0	0	0.81	- 0.03	0.74	- 0.01	- 0.16	0
Poi	0	4.83	0	- 0.05	- 0.05	0.86	0	0	0	1.08	-0.02	0.60	- 0.002	- 0.31	0
Xalan	0	- 0.01	0	0.02	- 0.04	0.75	0	0.12	0	1.83	- 0.02	0.49	- 0.01	0.04	0
Camel	0	6.41	0	- 0.12	- 0.05	0.41	0	0	0	1.08	- 0.01	0.19	0.01	- 0.95	0
Ant	0	7.99	- 0.01	0.03	- 0.06	1.24	0	0	0	0.87	- 0.02	0.50	- 0.01	- 0.39	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	PExcCRVD	PTCRV	PTCRVD	PConCRV	PConCRVD	PComCRV	PComCRVD	PSCV	PSCVD	WMC	DIT	NOC	CBO	RFC	LCOM	Intercept
Synapse	0	0	0	0	0	0	0	- 0.02	- 1.73	0	0	0	- 0.002	- 0.001	- 0.0001	1.7939
Velocity	0	0	0	0	0	0	0	- 0.02	- 1.62	0	0	0	- 0.002	- 0.002	- 0.0001	1.8502
Poi	0	0	0	0	0	0	0	- 0.03	- 0.69	0	0	0	- 0.002	- 0.003	- 0.0001	1.7503
Xalan	0	0	0	0	0	0	0	- 0.031	0	- 0.03	0	0	- 0.002	- 0.002	0.001	2.3886
Camel	0	0	0	0	0	0	0	- 0.03	-0.45	0	0	0	- 0.003	0.001	- 0.001	1.4162
Ant	0	0	0	0	0	0	0	- 0.025	- 1.59	0	0	0	- 0.003	- 0.001	- 0.0001	1.708

The model at system *i* in Table D.5 is given by the following equation.

$$Let\ P_i = B_{i,1} \times PSRV + B_{i,2} \times PSRVD + B_{i,3} \times PNCRV + B_{i,4} \times PNCRVD + B_{i,5} \times PPCICRV + B_{i,6} \times PPCICRVD + B_{i,7} \times PFCRV + B_{i,8} \times PFCRVD + B_{i,9} \times PMCRV + B_{i,10} \times PMCRVD + B_{i,11} \times PDSCRV + B_{i,12} \times PDSCRVD + B_{i,13} \times PExpCRV + B_{i,14} \times PExpCRVD + B_{i,15} \times PExcCRV + B_{i,16} \times PExcCRVD + B_{i,17} \times PTCRV + B_{i,18} \times PTCRVD + B_{i,19} \times PConCRV + B_{i,20} \times PConCRVD + B_{i,21} \times PComCRV + B_{i,22} \times PComCRVD + B_{i,23} \times PSCV + B_{i,24} \times PSCVD + B_{i,25} \times WMC + B_{i,26} \times DIT + B_{i,27} \times NOC + B_{i,28} \times CBO + B_{i,29} \times RFC + B_{i,30} \times LCOM + B_{i,31}$$

$$Then\ M_i = \frac{e^{P_i}}{1 + e^{P_i}}$$

Table D.6: Multivariate logistic regression models built based on CK metrics, metrics' coefficients (all systems)

	WMC	DIT	NOC	CBO	RFC	LCOM	Intercept
	1	2	3	4	5	6	7
Synapse	- 0.0364	0	0	- 0.0002	- 0.0125	0.0007	1.3058
Velocity	-0.0306	0	0	- 0.0002	- 0.0154	0.0008	1.3337
Poi		0	0	0.0019	0.0218	- 0.0003	- 1.454
Xalan	- 0.047	0	0	- 0.0012	- 0.0164	0.0014	1.6946
Camel	- 0.0341	0	0	- 0.0006	- 0.0163	0.0005	1.2061
Ant	- 0.0325	0	0	- 0.001	- 0.0133	0.0008	1.106

The model at release *i* is given by the following equation.

$$Let\ P_i = B_{i,1} \times WMC + B_{i,2} \times DIT + B_{i,3} \times NOC + B_{i,4} \times CBO + B_{i,5} \times RFC + B_{i,6} \times LCOM + B_{i,7}$$

$$Then\ M_i = \frac{e^{P_i}}{1 + e^{P_i}}$$

Appendix E: Correlation between Metrics

Table E.1: Spearman correlation coefficients of coding standard violations-based metrics for Synapse System

	PSRV	PSRVD	PNCRV	PNCRVD	PPCICRV	PPCICRVD	PFCRV	PFCRVD	PMCRV	PMCRVD	PDSCRV	PDSCRVD	PExpCRV	PExpCRVD	PExcCRV	PExcCRVD	PTCRV	PTCRVD	PConCRV	PConCRVD	PComCRV	PComCRVD	PSCV	PSCVD
PSRV	1.00	-0.42	0.67	-0.21	0.35	0.31	0.72	0.57	0.18	0.17	0.47	0.44	0.73	0.61	0.16	0.16	0.24	0.24			0.47	-0.08	0.96	-0.50
PSRVD		1.00	-0.27	0.64	-0.10	-0.06	-0.35	-0.15	-0.07	-0.07	-0.15	-0.11	-0.41	-0.24	-0.10	-0.10	-0.10	-0.10			-0.36	0.20	-0.41	0.98
PNCRV			1.00	0.27	0.06	0.03	0.36	0.22	0.03	0.03	0.23	0.22	0.35	0.23	0.04	0.04	0.08	0.08			0.26	-0.11	0.52	-0.38
PNCRVD				1.00	-0.26	-0.24	-0.29	-0.14	-0.10	-0.10	-0.15	-0.11	-0.34	-0.21	-0.11	-0.11	-0.11	-0.10			-0.20	0.17	-0.23	0.61
PPCICRV					1.00	0.99	0.18	0.07	0.23	0.23	0.13	0.10	0.20	0.12	0.15	0.15	0.18	0.18			0.06	-0.14	0.39	-0.11
PPCICRVD						1.00	0.15	0.06	0.20	0.20	0.10	0.08	0.17	0.10	0.14	0.14	0.16	0.16			0.04	-0.13	0.35	-0.07
PFCRV							1.00	0.92	0.00	0.00	0.18	0.16	0.35	0.24	0.13	0.13	0.23	0.23			0.37	-0.06	0.73	-0.40
PFCRVD								1.00	-0.04	-0.04	0.10	0.10	0.19	0.17	0.07	0.07	0.13	0.13			0.30	0.05	0.60	-0.18
PMCRV									1.00	1.00	0.19	0.17	0.17	0.13	-0.01	-0.01	0.16	0.16			0.07	-0.05	0.19	-0.08
PMCRVD										1.00	0.19	0.16	0.17	0.13	-0.01	-0.01	0.16	0.16			0.07	-0.05	0.19	-0.08
PDSCRV											1.00	0.99	0.22	0.12	0.14	0.14	0.10	0.10			0.11	-0.13	0.50	-0.18
PDSCRVD												1.00	0.19	0.10	0.12	0.12	0.08	0.08			0.09	-0.12	0.47	-0.14
PExpCRV													1.00	0.94	0.04	0.04	0.13	0.13			0.37	-0.10	0.71	-0.46
PExpCRVD														1.00	0.02	0.02	0.07	0.07			0.32	0.00	0.62	-0.28
PExcCRV															1.00	1.00	-0.02	-0.02			0.06	-0.06	0.18	-0.09
PExcCRVD																1.00	-0.02	-0.02			0.06	-0.06	0.18	-0.09
PTCRV																	1.00	1.00			0.05	-0.10	0.26	-0.11
PTCRVD																		1.00			0.04	-0.10	0.26	-0.11
PConCRV																			1.00					
PConCRVD																				1.00				
PComCRV																					1.00	0.74	0.48	-0.36
PComCRVD																						1.00	-0.06	0.22
PSCV																							1.00	-0.45
PSCVD																								1.00

Table E.2: Spearman correlation coefficients of coding standard violations-based metrics for Velocity System

	PSRV	PSRVD	PNCRV	PNCRVD	PPCICRV	PPCICRVD	PFCRV	PFCRVD	PMCRV	PMCRVD	PDSCRV	PDSCRVD	PExpCRV	PExpCRVD	PExcCRV	PExcCRVD	PTCRV	PTCRVD	PConCRV	PConCRVD	PComCRV	PComCRVD	PSCV	PSCVD
PSRV	1.00	-0.13	0.66	-0.02	0.38	0.38	0.74	0.64	0.30	0.30	0.58	0.52	0.77	0.69	0.15	0.15	0.25	0.25			0.66	0.40	0.97	-0.22
PSRVD		1.00	0.00	0.63	-0.14	-0.13	-0.16	0.01	-0.20	-0.19	-0.03	0.07	-0.22	-0.08	-0.03	-0.03	-0.11	-0.10			-0.28	0.00	-0.13	0.98
PNCRV			1.00	0.46	0.26	0.26	0.33	0.27	0.05	0.04	0.18	0.14	0.37	0.32	0.11	0.11	0.06	0.05			0.41	0.28	0.54	-0.13
PNCRVD				1.00	-0.13	-0.12	-0.20	-0.07	-0.21	-0.20	-0.14	-0.07	-0.22	-0.11	-0.03	-0.03	-0.15	-0.15			-0.12	0.11	-0.05	0.60
PPCICRV					1.00	1.00	0.30	0.18	0.17	0.17	0.18	0.13	0.29	0.20	0.11	0.11	0.21	0.20			0.20	0.04	0.39	-0.17
PPCICRVD						1.00	0.30	0.18	0.17	0.17	0.17	0.13	0.28	0.20	0.11	0.10	0.20	0.19			0.20	0.05	0.39	-0.16
PFCRV							1.00	0.94	0.19	0.19	0.47	0.41	0.54	0.45	0.15	0.15	0.24	0.23			0.43	0.18	0.76	-0.20
PFCRVD								1.00	0.10	0.10	0.40	0.39	0.41	0.39	0.12	0.12	0.15	0.15			0.35	0.21	0.68	-0.02
PMCRV									1.00	1.00	0.25	0.19	0.08	0.00	0.11	0.11	0.14	0.13			0.24	0.07	0.32	-0.20
PMCRVD										1.00	0.25	0.18	0.07	-0.01	0.10	0.10	0.13	0.12			0.24	0.08	0.32	-0.19
PDSCRV											1.00	0.97	0.35	0.25	0.09	0.09	0.18	0.17			0.28	0.12	0.61	-0.06
PDSCRVD												1.00	0.26	0.21	0.09	0.09	0.14	0.13			0.23	0.12	0.55	0.04
PExpCRV													1.00	0.94	0.07	0.07	0.21	0.20			0.55	0.30	0.75	-0.29
PExpCRVD														1.00	0.03	0.03	0.16	0.16			0.53	0.37	0.69	-0.14
PExcCRV															1.00	1.00	0.13	0.12			0.02	-0.03	0.16	-0.03
PExcCRVD																1.00	0.13	0.12			0.02	-0.03	0.16	-0.02
PTCRV																	1.00	1.00			0.03	-0.05	0.28	-0.11
PTCRVD																		1.00			0.02	-0.05	0.28	-0.10
PConCRV																			1.00					
PConCRVD																				1.00				
PComCRV																					1.00	0.88	0.65	-0.32
PComCRVD																						1.00	0.40	-0.02
PSCV																							1.00	-0.18
PSCVD																								1.00

Table E.3: Spearman correlation coefficients of coding standard violations-based metrics for Poi System

	PSRV	PSRVD	PNCRV	PNCRVD	PPCICRV	PPCICRVD	PFCRV	PFCRVD	PMCRV	PMCRVD	PDSCRV	PDSCRVD	PExpCRV	PExpCRVD	PExcCRV	PExcCRVD	PTCRV	PTCRVD	PConCRV	PConCRVD	PComCRV	PComCRVD	PSCV	PSCVD
PSRV	1.00	-0.04	0.69	-0.03	0.59	0.31	0.60	0.50	0.29	0.29	0.69	0.37	0.73	0.43	0.09	0.09	0.10	0.10			0.38	0.22	0.95	-0.18
PSRVD		1.00	0.10	0.72	0.10	0.35	-0.17	-0.02	-0.13	-0.12	-0.03	0.33	-0.09	0.33	-0.02	-0.02	0.02	0.03			-0.20	0.02	-0.05	0.97
PNCRV			1.00	0.43	0.32	0.14	0.40	0.37	0.13	0.13	0.37	0.18	0.39	0.23	0.03	0.03	0.00	-0.01			0.21	0.12	0.56	-0.06
PNCRVD				1.00	0.04	0.24	-0.15	-0.02	-0.14	-0.13	-0.09	0.21	-0.12	0.21	-0.04	-0.04	-0.12	-0.12			-0.15	0.02	-0.08	0.68
PPCICRV					1.00	0.87	0.13	0.07	0.09	0.09	0.31	0.18	0.26	0.15	-0.07	-0.07	-0.03	-0.03			0.06	0.01	0.59	0.03
PPCICRVD						1.00	-0.08	-0.08	0.00	0.00	0.12	0.19	0.07	0.18	-0.07	-0.07	-0.09	-0.09			-0.02	0.05	0.34	0.33
PFCRV							1.00	0.95	0.26	0.26	0.23	-0.06	0.31	0.02	0.10	0.10	0.11	0.11			0.46	0.31	0.59	-0.25
PFCRVD								1.00	0.18	0.18	0.14	-0.05	0.23	0.06	0.09	0.09	0.07	0.07			0.39	0.31	0.49	-0.10
PMCRV									1.00	1.00	0.18	0.00	0.13	-0.03	0.14	0.14	0.06	0.06			0.19	0.10	0.34	-0.14
PMCRVD										1.00	0.17	0.00	0.12	-0.03	0.14	0.14	0.06	0.06			0.19	0.10	0.34	-0.14
PDSCRV											1.00	0.80	0.50	0.32	0.04	0.04	0.07	0.07			0.18	0.05	0.70	-0.10
PDSCRVD												1.00	0.28	0.42	0.01	0.01	-0.05	-0.05			-0.06	-0.04	0.41	0.30
PExpCRV													1.00	0.77	0.01	0.01	0.06	0.05			0.26	0.12	0.70	-0.18
PExpCRVD														1.00	0.01	0.01	-0.04	-0.04			0.02	0.06	0.44	0.28
PExcCRV															1.00	1.00	-0.01	-0.01			0.07	0.05	0.09	-0.03
PExcCRVD																1.00	-0.01	-0.01			0.07	0.05	0.09	-0.03
PTCRV																	1.00	1.00			0.08	0.04	0.13	0.02
PTCRVD																		1.00			0.08	0.04	0.13	0.03
PConCRV																			1.00					
PConCRVD																				1.00				
PComCRV																					1.00	0.93	0.37	-0.22
PComCRVD																						1.00	0.22	0.01
PSCV																							1.00	-0.14
PSCVD																								1.00

Table E.4: Spearman correlation coefficients of coding standard violations-based metrics for Xalan System,

	PSRV	PSRVD	PNCRV	PNCRVD	PPCICRV	PPCICRVD	PFCRV	PFCRVD	PMCRV	PMCRVD	PDSCRV	PDSCRVD	PExpCRV	PExpCRVD	PExcCRV	PExcCRVD	PTCRV	PTCRVD	PConCRV	PConCRVD	PComCRV	PComCRVD	PSCV	PSCVD
PSRV	1.00	-0.17	0.53	-0.18	0.46	0.44	0.72	0.57	0.52	0.51	0.78	0.55	0.74	0.51	0.25	0.25	0.37	0.32	0.06	0.06	0.69	0.29	0.98	-0.25
PSRVD		1.00	0.07	0.65	-0.23	-0.17	-0.11	0.06	-0.16	-0.13	-0.19	0.12	-0.17	0.12	-0.06	-0.06	0.00	0.06	-0.01	-0.01	-0.11	0.28	-0.18	0.98
PNCRV			1.00	0.57	0.20	0.19	0.28	0.23	0.21	0.21	0.32	0.21	0.32	0.23	0.07	0.07	0.15	0.14	0.02	0.02	0.28	0.08	0.53	0.04
PNCRVD				1.00	-0.21	-0.17	-0.18	-0.04	-0.20	-0.18	-0.26	-0.03	-0.23	-0.01	-0.10	-0.09	-0.04	0.00	-0.01	-0.01	-0.21	0.06	-0.17	0.67
PPCICRV					1.00	0.99	0.15	0.07	0.23	0.22	0.34	0.15	0.33	0.17	-0.04	-0.04	0.06	0.03	-0.03	-0.03	0.28	-0.01	0.51	-0.25
PPCICRVD						1.00	0.14	0.08	0.20	0.20	0.31	0.16	0.30	0.17	-0.04	-0.04	0.05	0.02	-0.03	-0.03	0.26	0.02	0.49	-0.19
PFCRV							1.00	0.93	0.38	0.37	0.47	0.29	0.42	0.25	0.26	0.26	0.30	0.26	0.02	0.02	0.47	0.21	0.69	-0.16
PFCRVD								1.00	0.22	0.22	0.33	0.29	0.27	0.22	0.18	0.18	0.21	0.20	0.00	0.00	0.42	0.30	0.56	0.03
PMCRV									1.00	1.00	0.37	0.13	0.44	0.21	0.28	0.27	0.12	0.07	0.12	0.12	0.23	-0.03	0.52	-0.21
PMCRVD										1.00	0.35	0.13	0.43	0.21	0.27	0.27	0.10	0.06	0.12	0.12	0.22	-0.02	0.51	-0.19
PDSCRV											1.00	0.83	0.47	0.25	0.13	0.13	0.21	0.16	0.07	0.07	0.57	0.24	0.74	-0.27
PDSCRVD												1.00	0.24	0.22	0.06	0.06	0.09	0.08	0.05	0.05	0.47	0.39	0.52	0.07
PExpCRV													1.00	0.86	0.21	0.21	0.11	0.06	-0.01	-0.01	0.43	0.09	0.73	-0.24
PExpCRVD														1.00	0.11	0.11	-0.02	-0.04	-0.02	-0.02	0.30	0.18	0.51	0.08
PExcCRV															1.00	1.00	0.02	0.02	-0.01	-0.01	0.13	0.01	0.26	-0.09
PExcCRVD																1.00	0.02	0.02	-0.01	-0.01	0.13	0.01	0.26	-0.08
PTCRV																	1.00	0.99	0.03	0.03	0.24	0.11	0.39	-0.01
PTCRVD																		1.00	0.03	0.03	0.21	0.14	0.35	0.05
PConCRV																			1.00	1.00	-0.02	-0.03	0.06	-0.01
PConCRVD																				1.00	-0.02	-0.03	0.06	-0.01
PComCRV																					1.00	0.82	0.69	-0.14
PComCRVD																						1.00	0.29	0.27
PSCV																							1.00	-0.23
PSCVD																								1.00

Table E.5: Spearman correlation coefficients of coding standard violations-based metrics for Camel System

	PSRV	PSRVD	PNCRV	PNCRVD	PPCICRV	PPCICRVD	PFCRV	PFCRVD	PMCRV	PMCRVD	PDSCRV	PDSCRVD	PExpCRV	PExpCRVD	PExcCRV	PExcCRVD	PTCRV	PTCRVD	PConCRV	PConCRVD	PComCRV	PComCRVD	PSCV	PSCVD
PSRV	1.00	0.07	0.81	0.44	0.34	0.33	0.68	0.59	0.26	0.26	0.36	0.33	0.59	0.56	0.06	0.06	0.12	0.12			0.58	0.36	0.97	-0.03
PSRVD		1.00	0.07	0.44	-0.04	-0.03	-0.05	0.07	-0.05	-0.05	0.01	0.05	-0.11	-0.06	0.02	0.02	-0.05	-0.04			-0.22	-0.03	0.07	0.98
PNCRV			1.00	0.74	0.20	0.19	0.41	0.34	0.13	0.13	0.21	0.18	0.34	0.30	0.04	0.04	0.08	0.08			0.40	0.22	0.72	-0.06
PNCRVD				1.00	-0.01	0.00	0.10	0.16	-0.02	-0.02	0.05	0.06	0.02	0.04	0.02	0.02	0.01	0.01			0.08	0.12	0.40	0.36
PPCICRV					1.00	0.99	0.21	0.12	0.18	0.17	0.07	0.05	0.21	0.18	0.12	0.12	0.06	0.06			0.20	0.09	0.35	-0.07
PPCICRVD						1.00	0.20	0.12	0.18	0.17	0.07	0.05	0.21	0.18	0.12	0.12	0.06	0.06			0.19	0.09	0.34	-0.06
PFCRV							1.00	0.95	0.15	0.14	0.04	0.00	0.33	0.29	0.04	0.04	0.10	0.10			0.41	0.23	0.66	-0.12
PFCRVD								1.00	0.09	0.09	-0.01	-0.03	0.23	0.21	0.03	0.03	0.04	0.04			0.34	0.23	0.58	0.01
PMCRV									1.00	1.00	0.06	0.04	0.23	0.19	-0.01	-0.01	0.16	0.16			0.15	0.05	0.27	-0.07
PMCRVD										1.00	0.06	0.04	0.23	0.19	-0.01	-0.01	0.15	0.15			0.15	0.05	0.27	-0.07
PDSCRV											1.00	0.99	0.16	0.13	-0.02	-0.02	0.00	0.00			0.17	0.09	0.41	0.01
PDSCRVD												1.00	0.12	0.10	-0.02	-0.02	-0.01	-0.01			0.14	0.09	0.37	0.05
PExpCRV													1.00	0.98	0.07	0.07	0.11	0.11			0.32	0.12	0.62	-0.15
PExpCRVD														1.00	0.07	0.07	0.07	0.07			0.29	0.13	0.59	-0.09
PExcCRV															1.00	1.00	0.00	0.00			0.03	0.02	0.06	0.01
PExcCRVD																1.00	0.00	0.00			0.03	0.02	0.06	0.01
PTCRV																	1.00	1.00			0.05	-0.01	0.13	-0.04
PTCRVD																		1.00			0.05	-0.01	0.13	-0.04
PConCRV																			1.00					
PConCRVD																				1.00				
PComCRV																					1.00	0.92	0.58	-0.24
PComCRVD																						1.00	0.38	-0.04
PSCV																							1.00	0.01
PSCVD																								1.00

Table E.6: Spearman correlation coefficients of coding standard violations-based metrics for Ant System

	PSRV	PSRVD	PNCRV	PNCRVD	PPCICRV	PPCICRVD	PFCRV	PFCRVD	PMCRV	PMCRVD	PDSCRV	PDSCRVD	PExpCRV	PExpCRVD	PExcCRV	PExcCRVD	PTCRV	PTCRVD	PConCRV	PConCRVD	PComCRV	PComCRVD	PSCV	PSCVD
PSRV	1.00	-0.05	0.73	-0.06	0.40	0.38	0.78	0.50	0.40	0.39	0.49	0.41	0.79	0.58	0.17	0.17	0.57	0.52	0.04	0.04	0.63	0.16	0.97	-0.14
PSRVD		1.00	0.01	0.69	-0.11	-0.08	-0.14	0.18	-0.13	-0.12	0.05	0.15	-0.10	0.16	-0.04	-0.04	-0.15	-0.08	-0.02	-0.02	-0.04	0.32	-0.02	0.98
PNCRV			1.00	0.36	0.21	0.20	0.49	0.28	0.23	0.23	0.24	0.18	0.44	0.27	0.08	0.08	0.35	0.31	0.04	0.04	0.39	0.04	0.63	-0.12
PNCRVD				1.00	-0.15	-0.13	-0.21	0.00	-0.14	-0.13	-0.06	0.01	-0.20	-0.02	-0.06	-0.06	-0.21	-0.16	-0.02	-0.02	-0.11	0.13	-0.08	0.66
PPCICRV					1.00	1.00	0.20	0.02	0.22	0.21	0.14	0.09	0.32	0.18	0.07	0.07	0.22	0.18	-0.01	-0.01	0.19	-0.04	0.43	-0.12
PPCICRVD						1.00	0.18	0.02	0.20	0.19	0.13	0.08	0.30	0.17	0.06	0.06	0.20	0.16	-0.01	-0.01	0.18	-0.03	0.42	-0.09
PFCRV							1.00	0.82	0.27	0.26	0.30	0.21	0.53	0.32	0.15	0.15	0.44	0.38	0.02	0.02	0.50	0.09	0.75	-0.21
PFCRVD								1.00	0.05	0.06	0.11	0.10	0.24	0.23	0.05	0.05	0.17	0.17	0.00	0.00	0.40	0.28	0.50	0.13
PMCRV									1.00	1.00	0.18	0.11	0.37	0.19	0.10	0.10	0.16	0.11	-0.01	-0.01	0.18	-0.06	0.41	-0.16
PMCRVD										1.00	0.17	0.11	0.36	0.20	0.09	0.09	0.15	0.10	-0.01	-0.01	0.18	-0.05	0.41	-0.14
PDSCRV											1.00	0.98	0.34	0.22	0.19	0.19	0.20	0.16	0.06	0.06	0.22	0.02	0.53	0.02
PDSCRVD												1.00	0.26	0.19	0.15	0.15	0.13	0.11	0.05	0.05	0.18	0.06	0.46	0.13
PExpCRV													1.00	0.88	0.09	0.09	0.40	0.34	0.02	0.02	0.37	-0.04	0.77	-0.18
PExpCRVD														1.00	0.03	0.03	0.20	0.19	0.01	0.01	0.27	0.08	0.60	0.11
PExcCRV															1.00	1.00	0.12	0.08	-0.01	-0.01	0.08	-0.02	0.20	-0.04
PExcCRVD																1.00	0.11	0.08	-0.01	-0.01	0.08	-0.02	0.20	-0.04
PTCRV																	1.00	0.99	-0.02	-0.02	0.30	-0.03	0.57	-0.20
PTCRVD																		1.00	-0.02	-0.02	0.28	0.00	0.52	-0.13
PConCRV																			1.00	1.00	0.02	-0.01	0.05	-0.02
PConCRVD																				1.00	0.02	-0.01	0.05	-0.02
PComCRV																					1.00	0.79	0.64	-0.05
PComCRVD																						1.00	0.20	0.35
PSCV																							1.00	-0.08
PSCVD																								1.00

Appendix F: JPL Coding Standard

F.1 JPL Coding Standard's Rules Mapping to PMD, FindBugs and CheckStyle Rules

The mapping shown in Table F.1 is based primarily on the recommendations presented in the JPL standard document version 1.1 released in January 25, 2010. Those table's cells that marked with double strikes inside. Unfortunately, the recommendations does not cover all standard's rules may be because static analyzers did not support all standard's rules at the time of releasing it. This reason imposes us to spent more time and effort to review and investigate the whole set of rules provided by the most recent versions of the static analyzers we used to cover as much JPL standard's rules as possible. Those rules are shown in Table F.1 with a single strike mark.

Table F.1: The JPL standard's rules and their mappings to static the analyzers rules

JPL Rule	Checkstyle Check	PMD Rule	Findbugs Rule
"R01: compile with checks turned on."	--	--	--
"R02: apply static analysis."	--	--	--
"R03: document public elements."**	Method JavaDoc		
	Type Javadoc		
	Style javaDoc		
	Variable javaDoc		
"R04: write unit tests."	Junit test		
"R05: use the standard naming conventions."**	Abstract class names	ShortVariable	NM_CLASS_NAMING_CONVENTION
	Type names	LongVariable	NM_FIELD_NAMING_CONVENTION
	Constants names	ShortMethodName	NM_METHOD_NAMING_CONVENTION
	Local final	VariableNamingConventi	

	variable names	ons	
	Local variable names	MethodNamingConventions	
	Member names	ClassNamingConventions	
	Method names	AbstractNaming	
	Parameter Names	AvoidDollarSigns	
	Static variable names	MisleadingVariableName	
	class type parameter names	AvoidFieldNameMatchingTypeName	
	Method type parameter names	SuspiciousConstantFieldName	
		ShortClassName	
"R06: do not override field or class names."**	Hidden field	MethodWithSameNameAsEnclosingClass	
"R07: make imports explicit."**	Avoid star import	DontImportSun	
	avoid static import	DontImportJavaLang	
	illegal import	TooManyStaticImports	
	import order check		
"R08: do not have cyclic package and class dependencies."	--	--	--
"R09: obey the contract for equals()."**	Equals avoid null		BC_EQUALS_METHOD_SHOULD_WORK_FOR_ALL_OBJECTS
			EQ_ABSTRACT_SELF
			EQ_CHECK_FOR_OPERAND_NOT_COMPATIBLE_WITH_THIS
			EQ_SELF_NO_OBJECT
			NP_EQUALS_SHOULD_HANDLE_NULL_ARGUMENT
			EQ_UNUSUAL
"R10: define both equals() and hashCode()."**	Equals and hashcode	OverrideBothEqualsAndHashcode	HE_EQUALS_NO_HASHCODE
			HE_HASHCODE_NO_EQUALS
"R11: define equals			EQ_DOESNT_OVERRI

when adding fields.”			DE_EQUALS
“R12: define equals with parameter type Object.”**	Covariant equals		EQ_SELF_USE_OBJECT
“R13: do not use finalizers.”*	No finalizer	AvoidCallingFinalize	
R14: do not implement the Cloneable interface.*	No clone	CloneMethodMustImplementCloneable	
“R15: do not call nonfinal methods in constructors.”**		ConstructorCallsOverridableMethod	UR_UNINIT_READ_CALLED_FROM_SUPER_CONSTRUCTOR
“R16: select composition over inheritance.”			
“R17: make fields private.”**		SingularField	
“R18: do not use static mutable fields.”**		AssignmentToNonFinalStatic	MS_CANNOT_BE_FINAL
“R19: declare immutable fields final.”**		ImmutableField	
“R20: initialize fields before use.”**		DataflowAnomalyAnalysis	
“R21: use assertions.”**			NP_ARGUMENT_MIGHT_BE_NULL
“R22: use annotations.”**		FinalizeOverloaded	NP_BOOLEAN_RETURN_NULL
		SuspiciousHashcodeMethodName	NP_TOSTRING_COULD_RETURN_NULL
		SuspiciousEqualsMethodName	NP_NONNULL_RETURN_VIOLATION
“R23: restrict method overloading.”			
“R24: do not assign to parameters.”**	Parameter assignment	AvoidReassigningParameters	IP_PARAMETER_IS_DEAD_BUT_OVERWRITTEN
“R25: do not return null arrays or collections.”**		ReturnEmptyArrayRatherThanNull	PZLA_PREFER_ZERO_LENGTH_ARRAYS
“R26: do not call System.exit.”**		DoNotCallSystemExit	DM_EXIT
“R27: have one concept per line.”*	Multiple variable declaration	OneDeclarationPerLine	

“R28: use braces in control structures.”**	Need braces	IfStmtsMustUseBraces	
		WhileLoopsMustUseBraces	
		IfElseStmtsMustUseBraces	
		ForLoopsMustUseBraces	
“R29: do not have empty blocks.”**	Empty blocks	EmptyCatchBlock	UCF_USELESS_CONTROL_FLOW
	Empty statement	EmptyIfStmt	UCF_USELESS_CONTROL_FLOW_NEXT_LINE
		EmptyWhileStmt	
		EmptyTryBlock	
		EmptyFinallyBlock	
		EmptySwitchStatements	
		UncommentedEmptyMethod	
		UncommentedEmptyConstructor	
		EmptyFinalizer	
“R30: use breaks in switch statements.”**	Fall through	MissingBreakInSwitch	SF_DEAD_STORE_DUE_TO_SWITCH_FALLTHROUGH
			SF_DEAD_STORE_DUE_TO_SWITCH_FALLTHROUGH_TO_THROW
			SF_SWITCH_FALLTHROUGH
“R31: end switch statements with default.”**	Missing switch default	DefaultLabelNotLastInSwitchStmt	SF_SWITCH_NO_DEFAULT
	Defaults comes last	SwitchStmtsShouldHaveDefault	
“R32: terminate if-else-if with else.”			
“R33: restrict side effects in expressions.”**		AssignmentInOperand	
“R34: use named constants for non-trivial literals.”*	Multiple string literals	AvoidDuplicateLiterals	
	Magic numbers		
“R35: make operator			IM_MULTIPLYING_RE

precedence explicit.”*			SULT_OF_IREM
“R36: do not use reference equality.”**	string literal equality	CompareObjectsWithEquals	ES_COMPARING_PARAMETER_STRING_WITH_EQ
		UseEqualsToCompareStrings	RC_REF_COMPARISON_BAD_PRACTICE
			RC_REF_COMPARISON_BAD_PRACTICE_BOOLEAN
			RC_REF_COMPARISON
“R37: use only short-circuit logic operators.”**			NS_DANGEROUS_NON_SHORT_CIRCUIT
“R38: do not use octal values.”**		AvoidUsingOctalValues	
“R39: do not use floating point equality.”**		BadComparison	FE_FLOATING_POINT_EQUALITY
“R40: use one result type in conditional expressions.”*	Avoid inline conditional		
“R41: do not use string concatenation operator in loops.”**			SBSC_USE_STRINGBUFFER_CONCATENATION
“R42: do not drop exceptions.”**			DE_MIGHT_IGNORE
			DE_MIGHT_DROP
			REC_CATCH_EXCEPTION
“R43: do not abruptly exit a finally block.”**		ReturnFromFinallyBlock	
		DoNotThrowExceptionInFinally	
“R44: use generics.”**			BC_UNCONFIRMED_CAST
“R45: use interfaces as types when available.”**	interface is a type	LooseCoupling	
“R46: use primitive types.”*			BX_BOXING_IMMEDIATELY_UNBOXED
			BX_BOXING_IMMEDI

			ATELY_UNBOXED_TO_PERFORM_COERCION
			BX_UNBOXING_IMMEDIATELY_REBOXED
			DM_BOXED_PRIMITIVE_FOR_PARSING
"R47: do not remove literals from collections."			
"R48: restrict numeric conversions."**		StringBufferInstantiationWithChar	ICAST_IDIV_CAST_TO_DOUBLE
			ICAST_INTEGER_MULTIPLY_CAST_TO_LONG
"R49: program against data races."**			IS_FIELD_NOT_GUARDED
			VO_VOLATILE_REFERENCE_TO_ARRAY
"R50: program against deadlocks."**			UL_UNRELEASED_LOCK
			UL_UNRELEASED_LOCK_EXCEPTION_PATH
			SWL_SLEEP_WITH_LOCK_HELD
			TLW_TWO_LOCK_WAIT
"R51: do not rely on the scheduler for synchronization."			
"R52: wait and notify safely."**		UseNotifyAllInsteadOfNotify	NO_NOTIFY_NOT_NOTIFYALL
			UW_UNCOND_WAIT
			WA_NOT_IN_LOOP
"R53: reduce code complexity."**	Maximum file length	ExcessiveImports	
	Non Commenting Source Statements	UnusedImports	
	Npath	ExcessiveClassLength	

	Complexity		
	Boolean Expression Complexity	NcssTypeCount	
	Method count	TooManyFields	
	Maximum method length	TooManyMethods	
	Maximum parameters	ExcessivePublicCount	
	Cyclomatic Complexity	ExcessiveMethodLength	
	Avoid nested blocks	NcssConstructorCount	
	Class Fan Out Complexity	ExcessiveParameterList	
	Maximum line length	CyclomaticComplexity	
	Nested if depth	NPathComplexity	
	Nested try depth	AvoidDeeplyNestedIfStmts	
	Nested for depth	LawOfDemeter	
	Unused import	SwitchDensity	

F.2: Static analyzers plug-ins screen shoots in Eclipse IDE

Figure F.1: The eclipse plugin with FindBugs' perspective

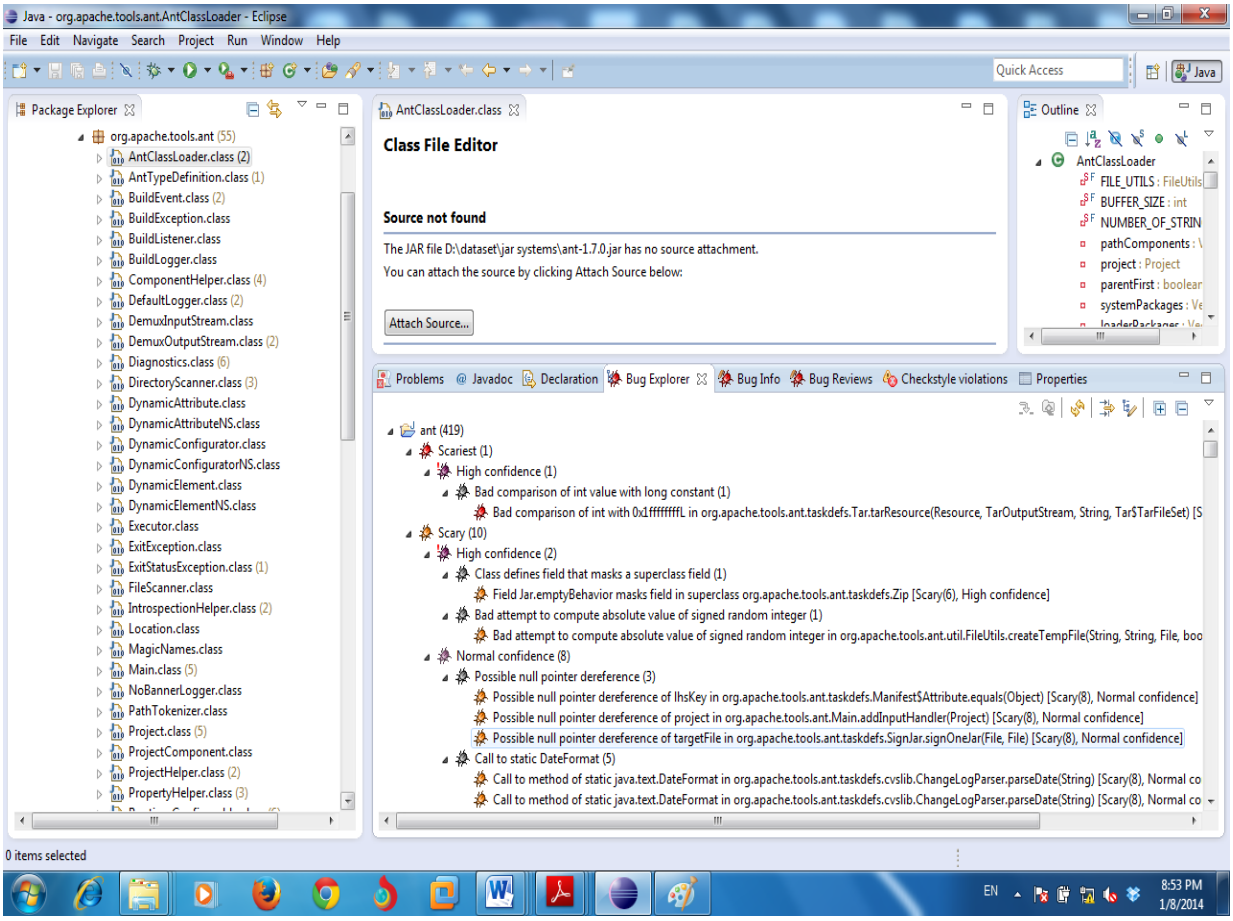


Figure F.2: The eclipse with plugin PMD's perspective

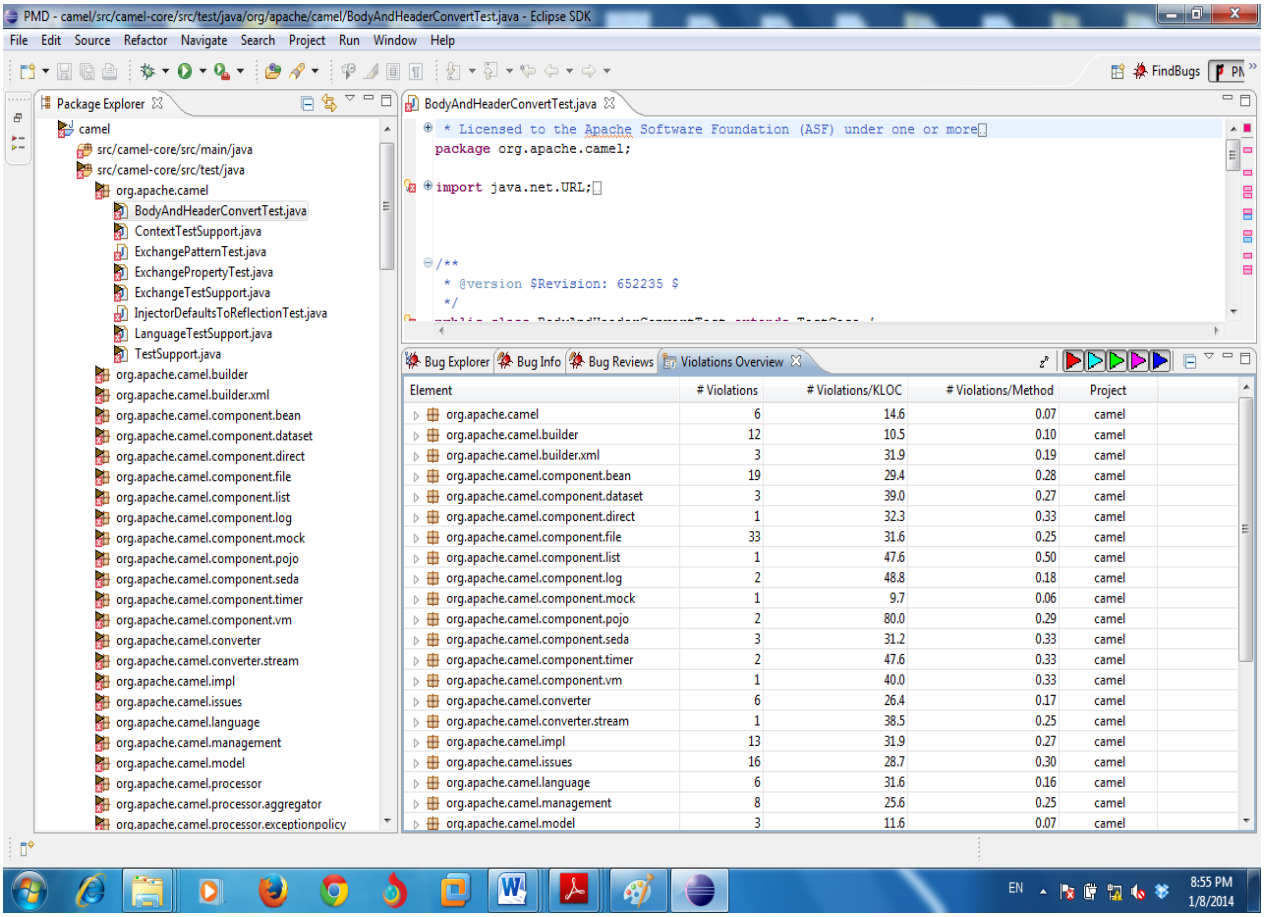
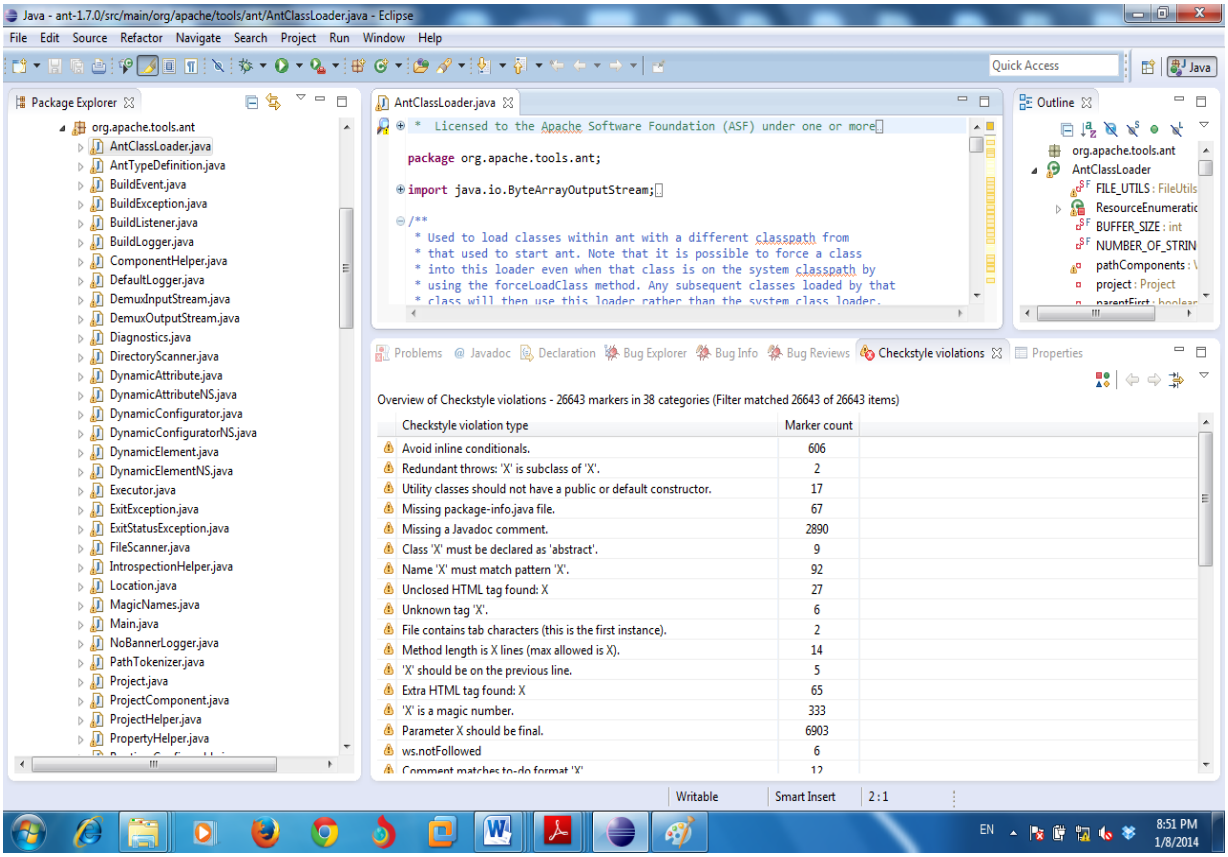


Figure F.3: The eclipse plugin with Checkstyle's perspective



Appendix G: The ER model for the coding rules violations database

The following figures present the ER Models for the coding rules' violations database, the entities with their attributes in addition to the relations between those entities are depicted.

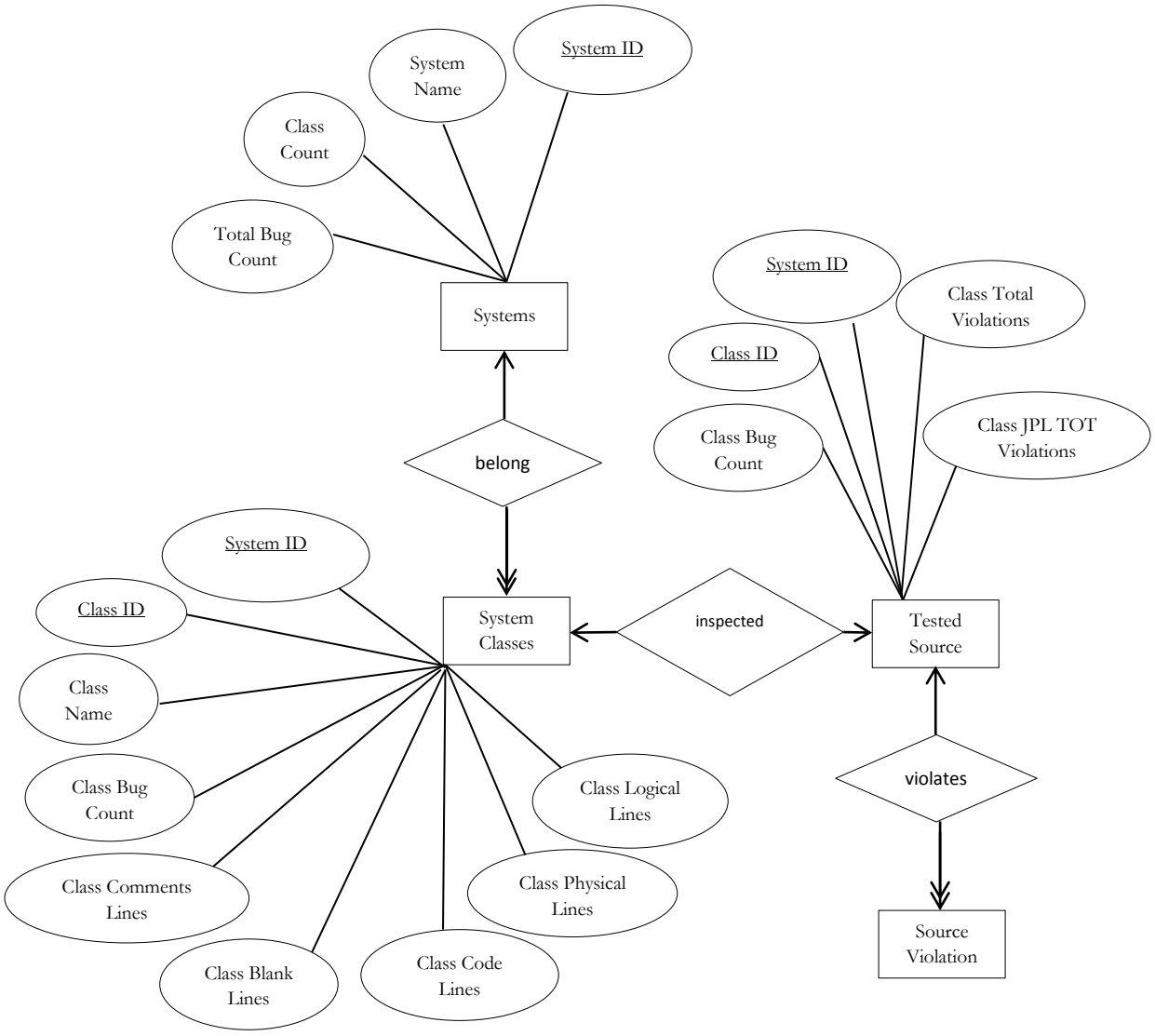


Figure G.1: The first part of the ER model

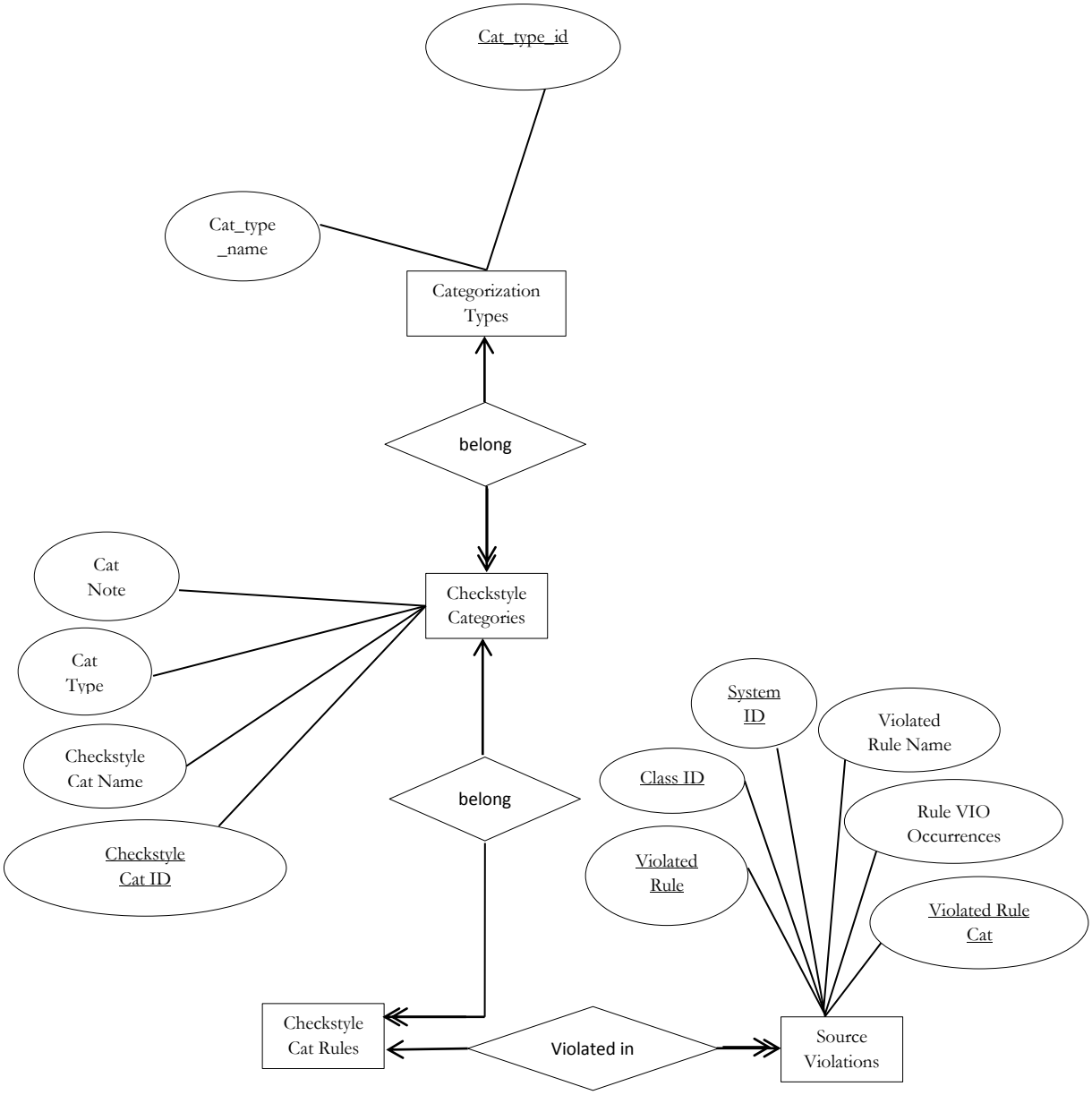


Figure G.2: The second part of the ER model

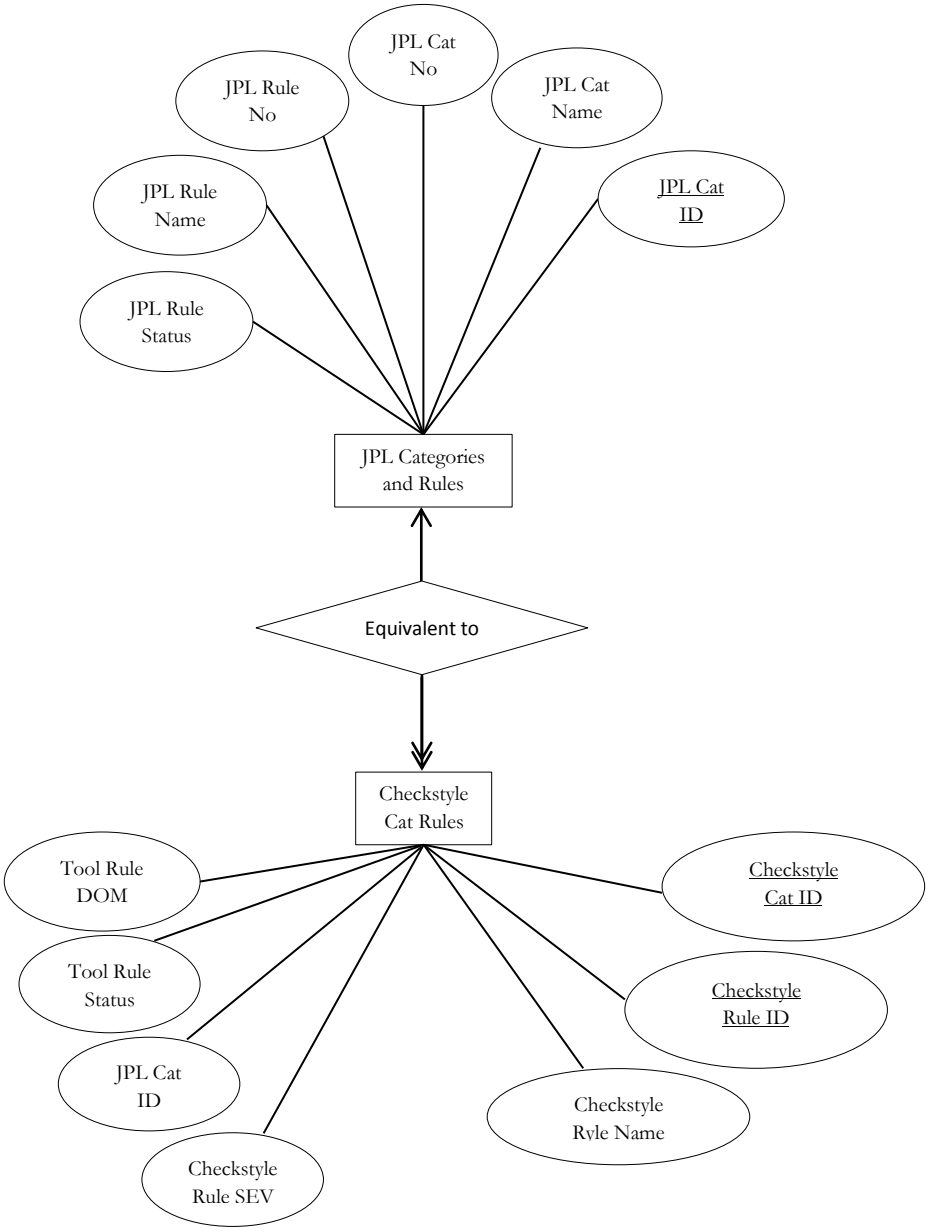


Figure G.3: The last part of the ER model

Vitae

Name : Bashar Qasem Hezam Ahmed

Nationality : Yemeni

Date of Birth :7/17/1981

Email : Bashar.Hozaim@gmail.com

Address : Taiz, Yemen

Academic Background : Received Bachelor of Science (B.Sc) in Computer Science from Taiz University in 2005 with GPA 85%